Error Propagation and Metamodeling for a Fidelity Tradeoff Capability in Complex Systems Design

A Thesis
Presented to
The Academic Faculty

by

Robert A. McDonald

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Aerospace Engineering
Georgia Institute of Technology
August 2006

Error Propagation and Metamodeling for a Fidelity Tradeoff Capability in Complex Systems Design

Approved by:

Professor Dimitri Mavris
School of Aerospace Engineering
Georgia Institute of Technology
Committee Chair

Professor James Craig
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Viren Kumar
General Electric Energy

Professor Alan Wilhite
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Michelle Kirby
Aerospace Systems Design Laboratory
Georgia Institute of Technology

Date Approved: July 2006

ACKNOWLEDGEMENTS

As with any substantial work of an individual, this one would not have been possible without the help and support of others. My life and this work have been touched over the years by too many people to mention. At the risk of an offense by omission, I humbly try to credit those most influential to this process.

Above all others, Dr. Mavris has made this work possible. His support and mentorship through my graduate education have provided both guidance and freedom. I must thank him for the opportunity to learn so much; and I must thank him for New Zealand.

The remainder of my reading committee, Dr. Craig and Dr. Kumar diligently served through my proposal and defense, providing invaluable suggestions, feedback, and guidance. They were joined in these duties by Dr. Wilhite and Dr. Kriby at the defense, where the entire committee were a credit to the academic community in their service. Dr. Kirby and Mr. Jeff Schutte deserve further thanks for their aid in developing the examples which demonstrate this work.

It is hard to describe how much support over the years has come from friends. Colleagues from ASDL provided encouragement and an endless supply of projects more interesting than my own. Groups of friends from the dog park and the Dixie Wing have provided escape and support in just the right doses. Brian served as consultant and sounding board for all things technical. Meanwhile, Claudia served as confidant for all things not. Amanda came into my life at just the right time to help me along and to convince me that the semicolon is not the new comma. Nick was a true friend through it all. And of course, I must thank my parents, who may be even happier to see this day than I am.

Thank you all,

Rob, July 2006

SUMMARY

Complex man-made systems are ubiquitous in modern technological society . The national air transportation infrastructure and the aircraft that operate within it, the highways stretching coast-to-coast and the vehicles that travel on them, and global communications networks and the computers that make them possible are all complex systems.

It is impossible to fully validate a systems analysis or a design process. Systems are too large, complex, and expensive to build test and validation articles. Furthermore, the operating conditions throughout the life cycle of a system are impossible to predict and control for a validation experiment. Sometimes, designers are interested in revolutionary systems for which there is no historical counterpart which can be used for validation.

Error is introduced at every point in a complex systems design process. Every error source propagates through the complex system in the same way information propagates. If a system has feedforward, the errors feed forward. If a system has feedback, then errors feed back. If a system has coupled loops, then errors are coupled.

As with error propagation through a single analysis, error sources grow and decay when propagated through a complex system. These behaviors are made more complex by the complex interactions of a complete system. This complication and the loss of intuition that accompanies it makes proper error propagation calculations even more important to aid the decision maker.

Error allocation and fidelity trade decisions answer questions like: Is the fidelity of a complex systems analysis adequate, or is an improvement needed? If an improvement is needed, how is that improvement best achieved? Where should limited resources be invested for the improvement of fidelity? How does knowledge of the imperfection of a model impact design decisions based on the model? How does this knowledge impact the choice and certainty of the design point? How does it impact the certainty of the performance of a

particular design?

In this research, a fidelity trade environment was conceived, formulated, developed, and demonstrated. This development relied on the advancement of enabling techniques including error propagation, metamodeling, and information management. These techniques were integrated with an existing commercial systems design framework and an intuitive graphical interface to create the fidelity trade environment.

A sensitivity approach to the propagation of error through complex systems was developed. This approach relied on the system sensitivity matrix to model the behavior of a complex system as a whole. In verification tests, the sensitivity approach provided approximate results substantially similar to a Monte Carlo approach which was many orders of magnitude more expensive. The rapid sensitivity approach to modeling error propagation enabled the rapid analysis required for an interactive environment.

A Gaussian process metamodel was used as an accurate surrogate of the component models which constitute the complex system model. The Gaussian process provided good approximation of the component responses and their derivatives, no matter how complex, throughout the design space, no matter how large. A novel interface was developed for training the Gaussian process and the metamodel was tightly integrated into an existing systems design environment. These advances helped to eliminate some of the barriers slowing the acceptance of Gaussian process metamodels. The Gaussian process metamodel enabled the rapid systems analysis and error propagation calculations required for the fidelity trade environment.

A relational database was designed and implemented to manage the vast amount of data and metadata involved in a complex systems design process. This database provided standardization and allowed global access to design information. The information management capability provided by the database enabled creation of the fidelity trade environment and its integration with an existing systems design architecture.

In two case studies, notional transport aircraft were modeled in the fidelity trade environment. In the first case study, the system was decomposed and the fidelity trade environment was used to integrate the system. In the second case study, a monolithic aircraft

synthesis tool was used. Then, scenarios were described where a decision maker used the fidelity trade environment at the beginning of a complex systems design problem. Using the environment, the designer was able to make design decisions while considering error and he was able to make decisions regarding required tool fidelity as the design problem continues. These decisions could not be made in a quantitative manner before the fidelity trade environment was developed.

This research identified the need for a new complex systems design technique. A fidelity trade environment was conceived, identifying the need for advancement of three enabling techniques, error propagation, metamodeling, and information management. All of these techniques were integrated with an existing systems design architecture and an intuitive graphical interface, thereby creating the fidelity trade environment. This environment was applied to two representative complex systems, thereby demonstrating its effectiveness in providing a new capability to the designer.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS OR ABBREVIATIONS

Aircraft Parameters

σ	Density ratio.
θ	Temperature ratio.
$C_{D,0\ cr}$	Zero-lift cruise drag coefficient.
$C_{D,0\ sl}$	Zero-lift sea level drag coefficient.
$C_{L,max,ld}$	Maximum landing lift coefficient.
$C_{L,max,to}$	Maximum takeoff lift coefficient.
$C_{L,max}$	Maximum takeoff lift coefficient.
e_{cr}	Oswald efficiency factor at cruise.
e_{sl}	Oswald efficiency factor at sea level.
$kC_{D,0}$	Zero-lift drag coefficient factor.
$kC_{D,i}$	Induced drag coefficient factor.
$kC_{D,ld}$	Landing drag coefficient factor.
$kC_{D,to}$	Takeoff drag coefficient factor.
$kSFC$	Specific fuel consumption factor.
kW_{fsys}	Fuel system weight factor.
kW_{furn}	Furnishings weight factor.
kW_{fuse}	Fuselage weight factor.
kW_{gear}	Landing gear weight factor.
kW_{ht}	Horizontal tail weight factor.
kW_{sc}	Surface controls weight factor.
kW_{vt}	Vertical tail weight factor.
kW_{wing}	Wing weight factor.
l_{fuse}	Fuselage length (ft).
$P_{s,OEI}$	OIE initial excess power (ft/min).
$P_{s,cr}$	Excess power at cruise (ft/min).
$P_{s,sl}$	Excess power at loiter (ft/min).

$P_{s,sl}$	Initial excess power (ft/min).
$P_{s,toc}$	Excess power at top of climb (ft/min).
R	Design range (nmi).
S	Wing area (ft^2).
S_{LDG}	Landing field length (ft).
S_{TO}	Takeoff field length (ft).
$S_{wet,add}$	Additional wetted area (ft^2).
$S_{wet,fuse}$	Fuselage wetted area (ft^2).
$S_{wet,wing}$	Wing wetted area (ft^2).
SFC	Cruise specific fuel consumption $(lbm/hr/lbf)$.
T/W	Thrust to weight ratio.
W	Takeoff gross weight (lbf).
W/S	Wing loading (lbf/ft^2).
W_f	Fuel weight (lbf).
W_f/W	Fuel fraction.
W_p	Payload weight (lbf).
W_{eng}	Engine weight (lbf).
$W_{f,add}/W_f$	Fuel fraction not used during cruise.
$\AA R$	Aspect ratio.

Error Propagation

$\mathbf{I}$	Identity matrix.
$\mathbf{M}$	System sensitivity matrix.
$\frac{\partial f}{\partial x}$	Partial derivative.
$\frac{df}{dx}$	Total derivative.
∂f	Partial diferential.
df	Total diferential.
δf	Partial deviation.
Δf	Total deviation.

ϵ_f	Partial uncertainty.
ε_f	Total uncertainty.
ς_f	Partial variance.
σ_f	Total variance.

Gaussian Processes

$\mathbf{C}_n$	Covariance matrix.
$\mathbf{\Theta}$	Hyperparameter vector.
$\mathbf{k}$	Covariance vector.
$\mathbf{r}$	Length scale hyperparameter vector.
$\mathbf{y}_n$	Observed response vector.
θ_1	Overall scale hyperparameter.
θ_3	Independent noise scale hyperparameter.
$C(x_1, x_2)$	Covariance function.
n	Number of observations.

System Description

$\mathbf{f}$	System responses.
$\mathbf{f}(\mathbf{x})$	Complex system.
$\mathbf{g}_i$	Subsystem responses.
$\mathbf{g}_i(\mathbf{x}, \mathbf{g}_{k\neq i})$	Subsystem.
$\mathbf{x}$	System variables.
$g_{i,j}$	Individual subsystem response.
l	Number of subsystems.
m	Number of system responses.
m_i	Number of subsystem responses.
n	Number of system variables.

System Solution

ϵ	Tolerance.
$\frac{\partial \mathbf{r}}{\partial \bar{\mathbf{g}}_i^*}$	Derivatives of residuals.
$\bar{\mathbf{g}}_i^*$	Subsystem feedback response guess.
$\mathbf{f}^*$	System feedback responses.
$\mathbf{g}_i^*$	Subsystem feedback responses.
$\mathbf{r}$	Residual.
k	Relaxation coefficient.

CHAPTER I

INTRODUCTION TO SYSTEMS DESIGN

Complex man-made systems are ubiquitous in modern technological society. The national air transportation infrastructure and the aircraft that operate within it, the highways stretching coast-to-coast and the vehicles that travel on them, and global communications networks and the computers that make them possible are all complex systems. All of these systems were designed through deliberate action to solve specific problems.

The design of these systems is a tremendously complex task, one that is far more involved than a cursory investigation would reveal. The design of an aircraft stretches far beyond the determination of its exterior form. The controls, propulsion, fuel, cooling, hydraulics, avionics, etc. are all complex systems in their own right. Even components as mundane as the windows, tires, and seats present significant engineering challenges.

It is the interacting operation of these components and subsystems that determines the performance of the whole system. The design of complex systems is dominated by understanding and modeling these interactions in order to assemble a complete entity capable of solving a problem.

1.1 Design

> Design establishes and defines solutions to [*sic*] and pertinent structures for problems not solved before, or new solutions to problems which have previously been solved in a different way. [1]

The key tenet to this definition of design is that design is a purposeful act; it is not random or casual. There exists a need, and to design is to satisfy that need. Appropriately, many problem-solving activities may be considered design under this definition.

The problem which is the subject of a design is established by recognizing some deficiency in the environment. This deficiency may be a complete lack of capability or a capability

with inadequate performance (measured in terms of cost, safety, speed, efficiency, capacity, etc.). Design may be used to solve a wide variety of problems.

Even an abstract and vague example such as *people and goods are in one place while they need to be moved to another* has solutions that pervade our everyday life. A ten-speed road bicycle, a four-door sedan, or a wide-bodied transport aircraft all move people and goods from one place to another and all were designed as solutions to this problem. One immediately notices that these solutions are very different. They are different because the general problem statement given above was not complete. Differing requirements on the capacity, speed, and operational flexibility presented by the problem led designers to vastly different solutions. In order to start the design process, the recognized deficiency must be completely laid out in the problem statement. However, it is important to note that most of the freedom a designer has to address a general problem is eliminated by the specific problem statement. Making a problem statement too specific can be just as dangerous to the success of a design as leaving the problem statement too vague [2].

The process of refining a problem statement may be seen as establishing a hierarchy of problems. A problem that pervades modern life is *to move people and goods from one place to another*. The need for the transportation form to *be safe and economical* refines the problem. The need for the transportation to *be fast and available on-demand* further refines the problem. Rather than describing one problem, this hierarchy describes a family of related problems.

The process of design itself also establishes problems within this hierarchy. If the designer chooses to address a need with a business aircraft, the designer is then presented with the problem of propelling the aircraft. If the designer chooses to address the need for propulsion with a jet engine, the designer is then presented with the problem of keeping the components of that engine cool. Again, this hierarchy does not condense all of the world's transportation needs down to turbine bucket cooling, it is but one problem in the hierarchy that may come into play.

1.2 Systems

> A "system" comprises a complex combination of resources, integrated in such a manner as to fulfill a designated need. A system is developed to accomplish a specific function, or series of functions[...] [3]

This definition of a system makes it clear that man-made systems are the product of design. A system is a solution to an identified problem, arrived at by deliberate action. A system is a combination of capabilities brought together to solve a greater problem. It is this network of interactions that makes a system distinct from one of its components. These interactions may involve feedforward and feedback, coupling, constraints, competing objectives, interdependence, etc.

Much like a problem definition, systems may exist at nearly any level of abstraction. Systems may be composed into larger systems of systems, or broken down into smaller subsystems and even components. The hierarchy of systems parallels the hierarchy of problems. The abstract need for transportation has led to the global transportation system. The national air transport system and interstate-highway network with all the diverse vehicles that traffic them are systems, as is a business aircraft with its jet engine, as well as the engine's turbine cooling flow circuit.

1.3 Systems Design

From these definitions it is apparent that the act of describing systems is design, and systems are created by being designed. This immediately introduces a fundamental question: What is the difference between design and systems design?

While the product of design is a system, traditional design does not decompose the system into components, nor does it focus on the interactions of those components. Instead, the system is treated as a monolithic whole. Conversely, in systems design, the focus is on decomposing the system into constituent parts, and then on understanding the interaction and balance of those parts [4]. The emergence of systems design as an extension of the design discipline has seen the development of techniques for the study of system decomposition

and interaction. In some cases, these techniques have been implemented as computational tools tailored for systems design.

Which design process is at work is not always obvious; in fact, similar systems may be described through design or systems design. Early automobiles were designed by individual inventors engaged in a monolithic design process. Modern automobiles are designed by a team of engineers, led by a "chief engineer," engaged in a systems design process. The growth in complexity of systems has dictated the transition from design to systems design.

While a complete description of the systems approach to design is far too involved for this discussion, a brief step-by-step outline to a computer-based systems design process will give context for the remaining discussion [5, 6, 7]. The process described herein purposely omits the fundamental preliminary steps of defining the problem and choosing a candidate system to solve the problem; instead, it focuses on the later steps involved in the analysis and design of a particular system.

The first, and defining, step to systems design is to decompose the system into a series of components; this decomposition is often guided by component function rather than the physical manifestation of the components. In so doing, the interactions between the components are identified such that the outputs of one feed the inputs of another. Then, appropriate physical and mathematical models for each of the components are identified. A numerical implementation of the component mathematical model is then developed or selected. Finally, the models of the components are assembled in a systems analysis environment according to their interactions. Design space studies such as optimization may then be performed on the system model. In order to improve and accelerate design studies, surrogate approximations based on the numerical model may be developed and used in place of the numerical model. These approximate models of the model are frequently referred to as metamodels [8, 9]. The systems design process generates a tremendous amount of information both directly related to the design study at hand as well as information describing the system model and the way the study was structured and carried out.

The desire to use a proven, off-the-shelf computational analysis code often drives the selection of the physical/mathematical model for a component. Even so, it is important

to consider the mathematical model choice as separate from and prior to the numerical implementation of the model in the following discussion.

1.4 An Elementary System

It is useful to introduce the elementary system depicted in Figure 1 as an example. This figure is a form of a Design Structure Matrix (DSM) [10]. The system involves four coupled components: A, B, C, and D. Each component is surrounded by a solid box. Information flow is represented by lines emanating from the boxes, inputs through the top and bottom of a box, outputs through the sides. A dot on a line identifies a specific piece of information carried in that flow. In this manner, the components are treated as black boxes.

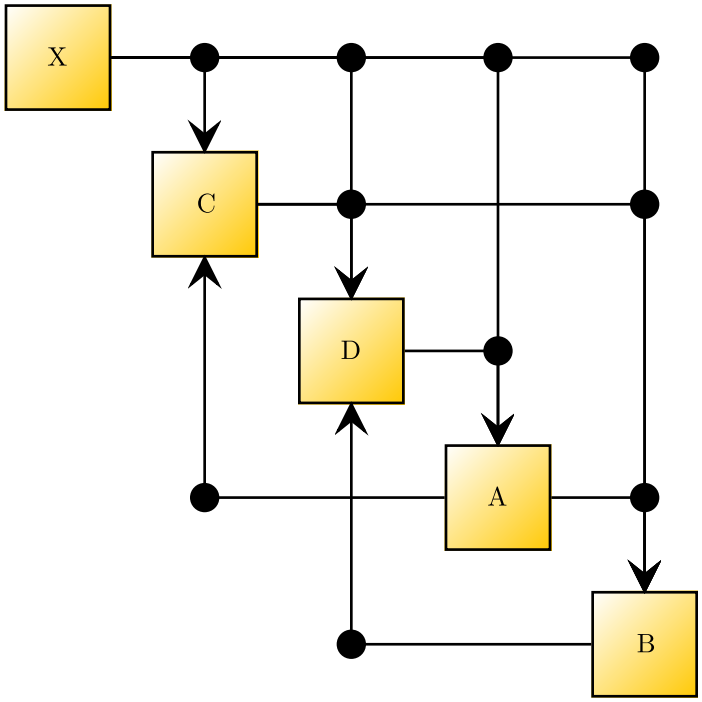

Figure 1: Elementary system Design Structure Matrix (DSM)

Together, these components form a system which is a function of an input vector, $X = \{x_i\}$, represented with another box. Each component produces an output vector, for example, analysis A produces the vector $A = \{a_i\}$. Each component may be a function of the input vector or the other components. The system outputs are an aggregation of the outputs of each component. In this example, A feeds forward to B and feeds back to C.

1.5 Challenges of Systems Treatment of Design

The systems perspective allows one to decompose and study a system at any level of abstraction using a set of techniques that are not specific to the problem at hand [11, 12]. Such techniques have been created to address the particular challenges of systems design: these techniques treat a system as a network of components with complex interactions and a component as a black box with inputs and outputs. In order to understand systems design techniques, it is important to understand the challenges each is intended to tackle. It is helpful to think of the example presented in Section 1.4 when considering the challenges of systems design discussed here.

1.5.1 System Behavior

The behavior of a system may present challenges to eventual design space exploration. Some problematic system traits include discrete inputs and highly nonlinear, discontinuous, or multi-modal outputs. Complex systems also often have multiple competing objectives, as well as nonlinear constraints restricting the design space. These traits pose challenges to optimization algorithms, visualization programs, and any other system level technique which relies on well-behaved functional behavior. Problematic system behavior may also be present at the subsystem or component level and directly leads to challenges for any metamodel that may be used.

1.5.2 Decomposition of Systems into Components

The decomposition of systems into interacting components introduces some challenges as well as great benefits. If the system is large and complex, many components with many interactions may result. These interactions may become difficult to track or understand. The optimal ordering or precedence of the components may become nonintuitive. Feedforward and feedback interactions can combine to form coupled loops requiring iteration to achieve consistency. Generally, the overall system is not equally sensitive to all of its components, and the relative importance of the components on the system is frequently not well understood.

1.5.3 Introduction of Physical and Mathematical Models of Components

The physical and mathematical models introduced for the components come with some unique challenges. The accuracy of the mathematical model may not be well understood or quantified. Further, the sensitivity of the component to its inputs observed in the overall variation of its outputs factors into the component's relevance to the system.

When the system components are modeled, frequently the inputs and outputs are not compatible in content. They may contain information at differing levels of abstraction, differing coordinate systems, differing base units, etc.

The introduction of physical and mathematical models also introduces some implicit constraints on the validity of the component analyses. Violation of these implicit constraints may be difficult to detect.

1.5.4 Introduction of Numerical Implementation of Component Models

The numerical implementation of the component models introduces another layer of challenges. The accuracy of the numerical method may be unknown. The computational costs of these components may be very different and some of the components may be computationally expensive.

Frequently, the inputs and outputs are not compatible in medium or format. Some use files while others use network or direct memory access as the medium for communication. Further, the communications format may be binary or plain text, it may be strictly formatted or free-form, or it may use a custom format or adhere to some standard.

Introduction of a numerical implementation also introduces some implicit constraints on the validity of the component analyses. The analysis code may fail to converge or give spurious results. Violation of implicit constraints may be difficult to detect.

Furthermore, certain numerical implementations may only be available on certain machines or certain computing platforms. The computers required for the complete system analysis may also not be in the same geographical location.

1.5.5 Information Management

The focus of systems design on the interaction of constituent parts of a complex system is in many ways an information management exercise. Unlike physical reality where interaction occurs through processes like contact, force, and heat transfer, the abstract models interact purely by passing information representing these processes. In addition to this overt data, the systems design process also involves a significant amount of implied data; this implied data is used to describe the overt data, and therefore is called metadata. Metadata describes the system, its decomposition, and the interactions between the constituents.

The metadata needed to describe a complex system describes the who, what, why, and how of the system model and all of the constituent models. This includes the domain of interest and the domain of validity, any assumptions and conditions implied or applied in the course of the study, and any information required for authentication, validation, and assurance of the data or metadata.

The data in a complex system is a record of the inputs and corresponding outputs of the contributing components. The data produced in a complex systems design study is typically of a much simpler structure than the metadata used to describe the system. However, the design study can produce huge quantities of data subject to rapid and frequent query.

Information management in a complex systems design study must provide a general structure capable of representing any complex system. The complex systems design process may be distributed across the globe requiring secure global access to the information.

1.5.6 Error and Error Propagation

Many of these aforementioned processes introduce error to the system analysis. Approximate operating conditions, materials properties, and other forms of input introduce error to the system. The iteration used to match coupled analyses introduces a convergence error. Any simplifications to the physics or approximations in the mathematical model introduce error. The numerical implementation of the model can introduce many forms of error, including iteration, discretization, and rounding error. Introducing a metamodel as a surrogate model introduces error. Error may also be present outside the system analysis. Any

design space exploration will have some error inherent in the final point decreed optimal.

The interacting nature of the components in the system leads to an interaction and flow of error. Error introduced at any stage propagates throughout the system and influences the final result. Depending on the relative importance and sensitivities of the components, the influence of a particular error source may grow or decay as it propagates throughout the system. It is very difficult, yet essential, to quantify the quality and accuracy of the final answer obtained through a complex systems design process.

Error can not be eliminated. However, it is possible to reduce many of the sources of error present in a complex systems analysis: better physical assumptions, fewer simplifications, better numerical implementations, and improved metamodels are all among the ways to reduce error at its source. Each error reduction has an associated cost while the complex system is varyingly sensitive to every error source. These relationships enforce a complex tradeoff between error sources. This tradeoff implies the ability to *allocate* an error *budget* associated with a complex system. For example, it does not make sense to converge a solution to within 10^{-6} when the physical assumptions are known to produce answers accurate to only 10%. There are currently no known methods or tools to help the designer balance the error budget; providing such capability is the primary aim of this research.

1.5.7 Validation of Systems

In most situations, it is impossible to fully validate a systems analysis or a design process. Systems are too large, complex, and expensive to build test and validation articles. Furthermore, the operating conditions throughout the life cycle of a system are impossible to predict and control for a validation experiment. Sometimes, designers are interested in revolutionary systems for which there is no historical counterpart which can be used for validation.

At present, systems analyses can not always be validated. Instead, each of the contributing component analyses are validated independently. Then, if each of the contributing analyses is trusted, it follows that the systems analysis built from them can be trusted. This can be called a build-up approach to systems validation.

Similarly, systems design and decision making techniques can not be validated. While it is possible to test decision making techniques on representative problems, because the actual conditions a system will encounter through its life cycle are unknown when it enters service, it is not possible to conduct controlled experiments simulating the life cycle of complex systems to be sure the right decision was made.

CHAPTER II

PROBLEM STATEMENT

Currently, the validation of systems analysis and design processes is not adequately addressed at a systems level. Instead, a build-up approach to validation is adopted where it is assumed that if the components in a systems analysis are validated and trusted, then when combined in a systems design process which has been tested and validated on small scale problems, the systems design process is valid.

However, this approach to systems validation does not address the flow, growth, and interaction of error through the complex system. Understanding the flow of error through a complex system is central to the buildup approach to systems validation.

Being aware of the shortcomings of a systems analysis process is only part of the struggle. Once aware of the impact of sources of error, the designer needs to be able to make fidelity decisions based on that awareness. Consequently, the systems model can be improved. These motivating observations lead to a summarizing research question which guides this work.

Research Question: *How can the designer better make fidelity decisions in a complex systems design process?*

Error allocation and fidelity trade decisions answer questions like: Is the fidelity of a complex systems analysis adequate, or is an improvement needed? If an improvement is needed, how is that improvement best achieved? Where should limited resources be invested for the improvement of fidelity? How does knowledge of the imperfection of a model impact design decisions based on the model? How does this knowledge impact the choice and certainty of the design point? How does it impact the certainty of the performance of a particular design?

Many tools have been developed to address the challenges of complex systems design.

These tools share a common purpose of trying to simplify the decision making process for the designer. This is best accomplished by presenting the designer with clear displays of the information required to make decisions. Once the designer has an environment for making decisions, he needs the ability to act on those decisions. Either the decision should be carried out, or the designer should be given the ability to see the impact of a decision in order to play what-if games.

In answer to the guiding research question, a new decision making capability which the designer can use to fully consider error is proposed; here, error is used as a measure of fidelity. The development of the ideas and methods implemented in the fidelity trade environment is the key focus of this research. The following hypothesis presents a testable answer to the research question.

Hypothesis 1: *If a fidelity trade environment is created, then when it is used in a complex systems design process, it will improve the designer's ability to make fidelity decisions.*

Of course, this hypothesis is built on a variety of definitions and concepts which have not yet been made clear. For example, it is not yet clear exactly what a fidelity trade environment is. A significant portion of this research is dedicated to understanding the problem of error in a complex systems design process, and thereby designing a tool to address that need.

The hypothesis given above is a testable answer to the guiding research question. The hypothesis must be tested through an experiment designed to support or disprove the hypothesis. To be meaningful, the experiment must be capable of proving the hypothesis false. Of course, passing an experiment does not prove a hypothesis true, only that it was not proven false by that experiment.

In order to test the hypothesis, a complex system must be modeled in the fidelity trade environment. The test system must have complexities representative of typical complex systems. The fidelity trade environment must then be exercised in a manner which demonstrates the use of the fidelity trade environment by a decision maker. The experiment not

only needs to demonstrate that the ideas and method embodied by the fidelity trade environment are possible and correct, but that the environment proves useful for the decision maker. Two primary experiments will be conducted to test this main hypothesis as well as further minor experiments to test other hypotheses presented later in the document.

CHAPTER III

SYSTEMS DESIGN TECHNIQUES

A multitude of techniques have been developed to address the unique challenges of systems design. These techniques have specific strengths and weaknesses inherent to their approach to the challenges of systems design. It is the purpose of this research to introduce a new systems design technique intended to address some of the challenges of systems design in a way that existing tools do not.

Fidelity analysis and decision making techniques establish a new class of systems design techniques. A fidelity trade environment is a decision making environment that combines system and fidelity analysis with design and fidelity decision making capabilities, thereby enabling the decision maker to simultaneously make design and fidelity decisions.

It is critical that a fidelity trade environment take a symbiotic place in the suite of systems design techniques already available to the designer. It must interoperate, using existing techniques to its advantage and being useful for other techniques. It is important to understand the context created by existing systems design techniques.

The hypothesized solution to the fundamental research question is a fidelity trade environment as a technique. Testing this hypothesis requires the implementation of a tool, but possible shortcomings of the tool do not necessarily invalidate the technique. It is hoped that future implementations of a fidelity trade environment will follow, each presenting additions to and refinements of the technique.

Systems design techniques include ideas, methods, and tools all meant to address the challenges of systems design laid out in Section 1.5. The relationship between ideas, methods, and tools and the way techniques evolve is important when developing a new technique.

Systems design techniques involve ways of solving a problem that may be discussed at various levels of abstraction: ideas, methods, and tools. Ideas refer to the most abstract and general of concepts; methods implement ideas as a well-defined approach; and tools

provide tangible implementations of methods.

For example, the observation that *nature does an excellent job of finding optimum solutions for complex problems* leads to the idea that designers should *mimic evolutionary optimization in systems design*. The advancement of this idea leads to the development of a method, a *genetic algorithm*. Furthermore, a corporation may choose to develop and market a specific tool say, *Acme's Super-GA*. Here, the idea, method, and tool are all considered techniques.

When considering the advancement of techniques, it is important to observe the mechanism by which that advancement takes place and how that mechanism varies depending upon the scope of the advance. Technique advances start with an idea. Ideas mature, are developed, and are tested until they become methods. Finally, the methods are implemented as tools.

In the case of this research, a new idea is pursued at the system level by developing a method and a tool. The research tool is an integrated, generic, proof of concept which couples well with an existing commercial general purpose umbrella tool. Comparisons to existing techniques in the literature will attempt to focus on ideas, methods, and tools as appropriate.

3.1 Taxonomy of Systems Design Techniques

Many of the techniques utilized by systems design may be divided into functional categories: design space exploration, decomposition and integration, visualization, and metamodeling. Together these techniques provide a powerful capability for addressing complex systems design. This research introduces a new category of systems design techniques, fidelity analysis and decision making.

Not all of the systems design challenges described in Section 1.5 are directly addressed by a particular systems design technique. For example, systems design techniques do not typically address the physical, mathematical, or numerical models used to represent a process. While choosing which model is important, the details and implementation are usually left up to the disciplinarian. However, the ramifications of those details are felt by the systems

level in a variety of ways such as error, computational cost and speed, input requirements, and input and output fidelity, etc.

3.1.1 Design Space Exploration

Design space exploration techniques transcend the challenges outlined for systems design; a challenging design implies a challenge to design space exploration.

Design space exploration techniques are concerned with efficiently coming to an understanding of the design space, and/or efficiently choosing a best point in the design space. These techniques include various forms of optimization, probabilistic analysis, and decision making techniques.

Any form of complex systems behavior will make design space exploration a challenge: nonlinearity, multimodality, discontinuous responses, and discrete inputs are all possible challenges. The sources and buildup of error through the system impacts the location, performance, accuracy, and robustness of the design point. The varied and unknown sensitivity of components to inputs, and of the system to components, presents challenges. Implicit constraints and ranges of validity of the underlying models introduce another class of challenges. Finally, the computational cost of the components must be considered for efficient design space exploration.

3.1.2 Decomposition and Integration

Decomposition and integration techniques are specifically aimed at the challenges introduced by decomposing systems into components [13]. These techniques establish the ordering and grouping of components. They also assume the burden of component compatibility and optimization, which has dramatic impact on the accuracy and efficiency of the systems analysis. These techniques ensure component interactions are properly handled.

Decomposition and integration techniques break the system into manageable components, interface these components to one another, convert outputs of one component into inputs of another, reorder components to best reconstruct the system, and provide the computing environments used to model the system.

When addressing component interactions, the compatibility and consistency of variables

in content and format is challenging. The introduction and quantification of error from all sources must be tracked by the system integration tool, as must the propagation and allocation of the error through the system. All modes of component analysis failure must be handled: this includes code crashes, silent failures, and violation of implicit ranges of validity. The realities of using diverse and distributed computing resources in parallel also puts forth challenges to these techniques.

Decomposition and integration techniques typically assume part of the information management burden. This usually includes the management of metadata and sometimes the management of data. All too frequently, the management of metadata is not supported by any tools and is left to the user. This usually results in an ad hoc approach to metadata management.

Error propagation and allocation are fundamental yet often overlooked aspects of complex systems integration. Rather than comprehensively track and understand error throughout the system, error is frequently mitigated locally while its growth or decay through the system is ignored; in extreme cases, error is ignored entirely. Comprehensive tools for propagation and allocation of error will improve design fidelity and confidence, reduce design cost, and will provide guidance for investment in analysis capabilities.

3.1.3 Visualization

Systems design visualization exists to elucidate complex systems behavior. Subsystems and components have complex behavior and interact in non-intuitive ways yielding system level behavior that is very difficult to understand. Systems design visualization affords the designer or decision maker confidence in his design or decision. Not only is insight into the system behavior obtained, insight of the component and system behavior may be used as a check to ensure the analysis was performed correctly. Visualization allows a great amount of information to be processed. This information is processed much more efficiently, and much more accurately, than would be possible by direct inspection of the data [14].

Systems design visualization techniques intuitively display the vast amounts of multivariate information inherent to systems design. This information may include system or

component behavior; the influence of constraints; and the creation, propagation, and impact of error. It is often desirable to make these information displays dynamic and interactive.

Visualization is primarily a mechanism for understanding the complex systems behavior or the behavior of components. Any challenge in system behavior may become an important reason to use visualization, and may present challenges to visualization. The computing cost of obtaining the data required for visualization can be great. However, these costs may be minimized if appropriate design space exploration or metamodeling activities are carried out.

3.1.4 Metamodeling

The use and acceptance of metamodeling as an engineering design tool has grown since its inception in the early 1970's [8, 9]. Metamodeling is intended to accelerate the ability to perform design space exploration or visualization by using limited information about the space to model the behavior of the space. This can speed the design process, allowing the designer a wider variety of designs and technologies. This acceleration also facilitates new approaches to design including probabilistic design and robust design. Metamodeling is also sometimes used to encapsulate expertise. A code which is difficult or complex to use may be modeled such that the difficulty may be addressed by an expert and the knowledge gained may be used in a simplified form. This encapsulation also allows some protection of intellectual property between collaborating organizations; an organization can share a capability without sharing the underlying tool.

In a metamodel, sample points are used to infer behavior between the sample points. Metamodeling techniques are concerned with the formulation, calibration, and implementation of surrogate models. The choice of points used for calibrating a metamodel is related to the design space exploration field; the main difference is that points are selected to reveal information about the design space rather than to reveal an optimum design. Metamodeling also involves characterizing and understanding the approximation inherent to the surrogate as well as assessing the quality of the resultant metamodel.

Any sort of challenges in the component or system behavior present challenges to metamodeling. Any significant random error introduced by the numerical method presents difficulties to the metamodels. Crashed cases present challenges. Metamodeling also introduces approximation error that should be quantified and understood throughout the system.

Metamodeling is frequently viewed as an optional aspect of systems design used to accelerate the design process; it is often dismissed as a stopgap measure until computer power catches up to software. Of course, physics-based analysis will always push the limits of computing power. Because of this general attitude, metamodeling has not been tightly integrated with the design environment and some of metamodeling's greatest strengths have gone untapped.

When metamodels are used, managing the sample points becomes a central part of information management. Again, frequently the management of this data is not supported by the tools and is left to the user. This usually results in an ad hoc approach to data management.

3.1.5 Fidelity Analysis and Decision Making

Fidelity analysis and decision making represents a new category of systems design techniques directly targeted at the challenges of error propagation and validation for complex systems. Like design space exploration techniques, fidelity analysis and decision making techniques must address all the challenges of systems design.

By analyzing the flow of error through the system fidelity analysis quantifies the build-up approach to system validation. Decision making tools built on fidelity analysis give the designer the ability to make fidelity decisions in the design process as well as make design decisions with an understanding of the fidelity of the system model.

Fidelity analysis and decision making techniques face all of the challenges of systems design. The propagation of error through a system faces all of the challenges of information flow through a system. An interactive environment requires rapid systems analysis with the consideration of error. The visualization challenges of systems design are compounded by adding additional degrees of freedom to every quantity in the system. Similarly, the

introduction of fidelity makes the decision space available to the designer larger and more intricate.

3.2 Fidelity Analysis and Decision Making Enablers

The proposed existence of a fidelity tradeoff environment suggests traits which may be cast as requirements for developing this new systems design technique. Some of these enabling techniques are outlined in this section. The exact implementation of these enablers is less critical to the success of the overall technique than is achieving the base capability that the enabler is meant to provide.

The development of an error tradeoff environment will rely on certain capabilities of existing systems design techniques including visualization, metamodeling, and information management. Some of these requirements are outlined in the following discussion.

Error analysis and visualization on a systems level will involve repeated systems analyses which must be completed rapidly enough to maintain an interactive feel for the user. This demands rapid system modeling and rapid modeling of the propagation of error through the system.

In order to compare one source of error to another, the relative contribution of each source of error to the propagated total error must be calculated. This is complicated by the fact that multiple sources of error do not combine through straightforward superposition.

Hypothesis 2: *The propagation of error through a complex system can be modeled efficiently, including the isolated impact of individual error sources.*

Rapid system modeling demands effective metamodeling. Complex systems can include components with difficult to model behavior. The range of interest modeled by these components and thereby the system should not be limited by the metamodel. As a component of the fidelity tradeoff environment, the metamodel must interoperate with existing systems design techniques.

Hypothesis 3: *Metamodels providing for the efficient approximation of arbitrarily complex functions and their derivatives on arbitrarily large domains can be created and integrated*

into a complex systems design environment.

There must be a centralized data repository for all data. As metamodel training cases are run, results go directly to the repository. A repository will prevent data from being discarded during the design process; data from related studies will be reused. In contrast to standard metamodeling practice, data will be hoarded in recognition of its intrinsic value; data will not be thrown away.

The data repository must store metadata about the design study and the system and component models. This information will document the analysis and design processes by recording all decisions and assumptions made. The metadata will also provide facilities for version tracking, accountability, validation, protection, and assurance.

There must be remote access to the data repository. With this capability, entities throughout the enterprise may contribute to and benefit from the up-to-date knowledge base encouraging cooperation and efficient use of resources. Without remote access, distributed groups within an enterprise would be forced to duplicate effort and capability.

Hypothesis 4: *Complex systems design information can be stored in a comprehensive, standardized, and centralized way.*

A new environment must seamlessly integrate with existing systems design environments. In this way it will be easy to use and adopt and it will leverage all the existing innovation, experience, and research going into systems design. Conversely, if the new system does not interoperate with existing systems, every needed feature must be reimplemented and experience must be regained.

The implementation of any new capability must not restrict the computing environment any more than the design environment it is being integrated into; it should be portable across any computing platform and should be built as much as possible in a modular way from off-the-shelf components. This will reduce development and testing time and makes established and powerful tools available, lending instant credibility to the tool.

There must be an interactive and dynamic setup capability for building the representation of design studies and identifying available resources. The inputs and outputs of the

wrapped analyses can be complex; a system that can interactively manage this for the user is required.

There must be an interactive and dynamic visualization capability. Visualization affords greater understanding of the workings of the system, builds confidence in the analysis, and yields design insight. Obtaining an intuitive understanding of the large volume of multivariate design information would be very difficult otherwise. A graphical representation of the design space is critical to making a decision with confidence. The complex role of error and its system-wide effects demand new visualization techniques.

There must also be an interactive error management capability. Most sources of error can not be estimated from only the information available to the systems design environment; instead, outside error estimates must be incorporated. A dynamic error management interface will provide a means for the designer to understand, balance, and budget error. This information will guide the selection and investment in analysis capabilities as well as traditional design decisions and provide means for the designer to explore the consequences of his decisions.

Decision making tools for complex systems design must delicately balance user interaction and automation. The temptation exists to build algorithms and heuristics to allow the computer to make decisions for the designer. Doing so may marginalize, alienate, and discourage the designer. Instead, design tools should support and compliment the designer and his natural workflow. This is best done by using the computer to process and display the information required to make decisions. Decision making tools should display only the decision relevant information, while hiding the underlying complexities of the system from the designer. Additionally, decision making tools should provide intuitive mechanisms for the designer to act on his decisions. Either the desired action should be carried out in a simple manner, or the impact of the decision should be modeled and reflected in the decision making tool.

The fidelity analysis and decision making techniques proposed in this research will improve the accuracy of design studies while reducing computation cost and time to complete the studies. Error will be estimated and its impact will be tracked and controlled throughout

the complex system. Error allocation will guide the efficient use of resources and future investment in analysis capability. Distributed groups will be given concurrent access to design space knowledge fostering cooperation and reducing cost by eliminating duplicated effort. Furthermore, this integrated technique will bring these enhanced benefits by interoperating with an existing systems design environment.

3.3 Document Roadmap

Section 3.2 outlined a set of enabling requirements for the creation of a fidelity trade environment. The remainder of this document is divided into three main parts as depicted in Figure 2; this diagram depicts the chapter-by-chapter flow and structure of this document. Each chapter is represented by a labeled box and appendices containing material supplementary to some chapters are depicted as smaller boxes attached to the appropriate chapter.

The first two remaining parts of the document follow parallel paths through the creation of a fidelity trade environment; these parallel paths are depicted by the grey boxes down the sides of Figure 2. The first of these parts discusses the traits, characteristics, and merits of the techniques which enable a fidelity trade environment. The second of these parts discusses the specific implementation choices for each of the techniques which enable a fidelity trade environment.

The remainder of the document is dedicated to two primary example applications of the fidelity trade environment and to a concluding chapter which summarizes the research performed and provides direction for the future. These chapters are grouped in the grey box at the bottom of the figure as the fidelity trade environment proof of concept.

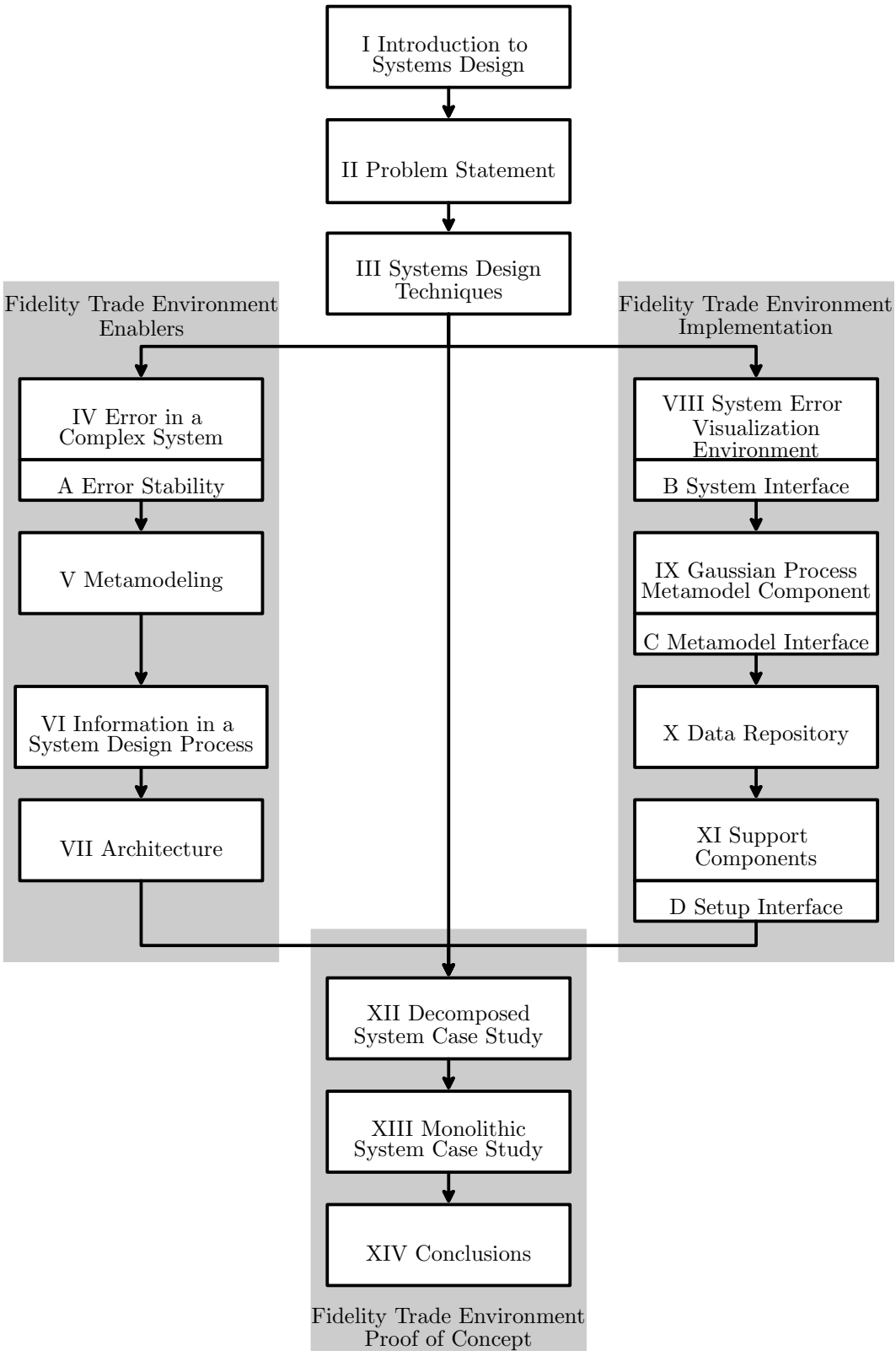

Figure 2: Document roadmap

CHAPTER IV

ERROR IN A COMPLEX SYSTEM

Error and uncertainty are ubiquitous in any complex systems study. As systems analysis and design techniques have progressed, there has been increasing interest and focus on the role of uncertainty in the system's life cycle and its impact on a robust design. Meanwhile, the role of error in the systems analysis and design processes has largely been neglected. In fact, the fundamental difference between error and uncertainty is not frequently recognized. This research does not focus on robust design, uncertainty propagation, error identification, or error quantification. Instead, this research focuses on error propagation for the purpose of error allocation and fidelity trades.

4.1 Verification and Validation

Any discussion of the use of tools in engineering should include a discussion of the related concepts of fidelity, error, validation, and verification. Roache [15] establishes technical definitions for the concepts of validation and verification in the context of analysis tools. In his structure, validation is a process of ensuring that one is solving the right equations and verification is a process of ensuring that one is solving the equations right.

In practice, this means that verification entails an objective test demonstrating that the analysis tool correctly generated a solution to the desired equations. Conversely, this means that validation entails a subjective test demonstrating the appropriateness of a certain set of equations to a particular problem. Demonstrating that an CFD code solves the Euler equations properly is a verification exercise. Determining whether the Euler equations provide a satisfactory approximation to the true behavior of the system of interest is a validation exercise.

While formal processes for verification exist for some classes of problems, often the verification of a tool is implied by a validation exercise. Typically, validation entails comparing

the result of a tool to some well-known benchmark solution or an experimental measurement.

In this research, the term fidelity is used as a measure of error. A tool with higher fidelity is defined to have less error. In this context, a judgement of fidelity has nothing to do with the sophistication of the tool, only of the error in the result. A validation exercise can be interpreted as quantifying the fidelity or error of a tool and then making a decision as to whether that level of fidelity is appropriate for the problem at hand. In this manner, a fidelity trade environment is a decision making tool for systems analysis validation.

Some verification experiments have been conducted throughout the process of this research and are included in this document. These experiments are designed to demonstrate the correctness of some of the components contributing to this work.

4.2 Error vs. Uncertainty

Error is a form of uncertainty. However, it is not appropriate to treat error the same as other kinds of uncertainty. In order to distinguish error from other kinds of uncertainty, an arbitrary semantic distinction is made between *error* and all other kinds of uncertainty, here called *uncertainty*. Some confusion is risked by establishing this subtle distinction rather than introducing a term for other kinds of uncertainty; however, this use of the term uncertainty most closely matches common use in uncertainty propagation and robust design. A complete taxonomy of all possible kinds of uncertainty will not be attempted here; however, effort will be made to identify the distinguishing features of error and uncertainty and to explain the differences in their use.

The concept of error propagation is closely tied to the concept of uncertainty propagation. In complex systems, uncertainty propagation has been studied in the context of robust design by various authors including Chen et al. [16], Du and Chen [17], Mavris and Bandte [18], and DeLaurentis and Mavris [19]. Robust design is concerned with finding a point within the design space that is insensitive to uncertainty [20]. In robust design, the sources of uncertainty must be well understood. Typically, a detailed description of the statistical distribution of the uncertainty of a quantity is needed. For example, the price of fuel has

uncertainty with an associated statistical distribution. This distribution has a shape (e.g., normal) with certain parameters (e.g., mean and standard deviation). Of course, for any given fuel purchase, the price paid is exactly known.

If one is performing a design study where the result is sensitive to weather, fuel cost, and a market estimate, appropriate statistical distributions for the uncertainty in these quantities can allow the designer to use robust design techniques to find a design point relatively insensitive to these factors. In its most simple form, robust design uses a Monte Carlo approach to sample the input quantities from the specified distributions. The design analysis is performed for each sample point and a statistical distribution of the result is constructed. In order to mitigate computational cost and time, variations on this approach have been devised and implemented. Sometimes, metamodels are used as a fast surrogate for the design analysis. Other times, techniques like fast probability integration (FPI) are used to speed the exploration of the statistical domain [21].

The uncertainties studied in robust design have three defining traits. First, there is usually nothing the designer can do to change the uncertainty. The designer can not alter the weather, control future fuel costs, or demand a certain market for a product. Second, the statistical distributions of the uncertainty can sometimes be well known. Historical weather records, commodities futures models, and economic forecasts can establish the shape and parameters of the uncertainty distribution. When historical information is not available, or the distributions of interest require extrapolations or forecasts, less may be known about the distribution of uncertainty. A designer can not change an uncertainty distribution; he can only change his knowledge of it. Third, the magnitude of change associated with uncertainty is relatively large. These defining characteristics are summarized in the following list.

Defining characteristics of uncertainty

- Uncertainty can not be changed by the designer.
- Uncertainty distributions can sometimes be well known.
- Uncertainty involves relatively large changes in quantities.

Some forms of uncertainty fall beyond this rudimentary taxonomy. For example, depending on the interpretation, uncertain requirements in the early stages of design may or

may not fit this categorization as uncertainty. In one interpretation, the designer may have the ability to impact uncertain requirements, thereby violating the first point; similarly, it may not be appropriate to treat the unknown requirements as probability distributions, possibly violating the second point. In an alternate interpretation, the designer may wish to use probability distributions to model the variety of missions a vehicle will encounter throughout its life cycle. In this situation, requirements uncertainty most likely qualifies as uncertainty in the present taxonomy.

Error differs from the other sources of uncertainty in the three defining traits. First, there *are* things the designer can do to change the error. A low fidelity analysis code can sometimes be replaced with a higher fidelity code or the results of a tailor-made experiment. An input quantity known to a certain accuracy can be re-measured with better instruments. While an individual designer may not have the power or authority to reduce a given source of error, conceptually almost all sources of error may be reduced in some way. Second, the shape of an error distribution is not well known. Seldom does an engineer know that the error in a particular analysis technique follows particular distribution (eg. normal, Weibull, or lognormal). In fact, the validation of analysis tools is a challenging research field in its own right [15, 22] where the focus is on estimating confidence bounds, and there is not enough information to develop a detailed probability distribution for the error. Third, the magnitude of change associated with error is typically small. The contrasting defining characteristics of error are summarized in the following list.

Defining characteristics of error

- Error can be changed by the designer.
- Error distributions are not well known.
- Error involves relatively small changes in quantities.

The size differences between uncertainty and error deserves further attention. While no absolute statement about the magnitude of uncertainty or error can be made, uncertainty will usually be large in relation to error. For example, the seasonal uncertainty of temperature at a location is much greater than the error in an individual temperature measurement.

This statement on the relative magnitude of error and uncertainty is only needed to support the use of sensitivity techniques for the propagation of error. The use of sensitivity techniques may be inappropriate for large changes, depending on the nonlinearity of the system. As discussed later in this document, verification and validation experiments were conducted with satisfactory results for each of the example systems in the thesis; in these experiments, the results of the sensitivity approach to error propagation were compared to a Monte Carlo approach.

The defining differences between uncertainty and error result in fundamental differences in the activities that can and should be conducted when considering these phenomena. While it may be interesting to know what impact a less uncertain fuel cost would have on a design, the designer has no way to cause the fuel cost to be less uncertain. On the other hand, the designer may choose to improve the accuracy of a tool in response to a what-if decision. Consequently, tools supporting tradeoff decisions make sense in the context of error, but not in the context of uncertainty for robust design.

In order to perform the what-if trades implied by error allocation, a very rapid analysis of the error propagation must be performed. The error magnitude is known to be small, but detailed knowledge about the distribution of error is not known. Due to these facts, it makes sense to perform error propagation based on sensitivity techniques. On the other hand, in robust design, detailed probability distributions are known. The uncertainty magnitude may be large and the distribution is beyond the control of the designer. Consequently, it makes sense to perform a more detailed analysis of the propagation of uncertainty by using a Monte Carlo method or related technique as discussed above.

There may be situations where one may want to perform a robust design on a system with error propagation. In that event, a hybrid of the two techniques may be constructed. A hybrid approach may consist of a detailed Monte Carlo approach with simple distribution shapes assumed for the error quantities. Alternatively, a sequential approach may be applied where sensitivity techniques are applied to the error propagation and Monte Carlo techniques are applied to the robust design variables. In either case, it is very likely that this type of study will not make sense in general practice. In most cases, the size of the

distributions investigated in robust design are far larger than the confidence bounds used by error propagation. This would result in the contribution of variation due to error being overshadowed by the contribution due to uncertainty.

Similarly, one may be tempted to use the sensitivity based error propagation techniques to investigate the impact of a small change in an individual variable. This is an appropriate use of sensitivity techniques which corresponds exactly to the construct of deterministic error. The statistical techniques used for random error would correspond to a situation where a designer knew the approximate magnitude of a potential design variable change, but did not know its exact magnitude or anything about its direction. This is most likely not what the designer wants when performing a sensitivity study. Instead, this sort of sensitivity analysis is better performed in the interactive design environment provided by tools the designer already has such as a standard sensitivity analysis or a design space exploration interface.

4.2.1 Illustration

In order to clarify the differences between uncertainty and error as defined here, and the differing use of robust design and fidelity trade techniques, a simple example is presented in familiar terms.

Imagine a model mounted in a wind tunnel. The tunnel is open to the atmosphere and has only one power setting. The model angle of attack can be adjusted over a range of values. The tunnel is equipped with a balance to measure aerodynamic forces and the instrumentation needed to record the operating conditions. Imagine also an analysis tool for simulating the experiment.

On a given day, an experiment was carried out in the tunnel. On that day, the temperature, pressure, tunnel velocity, model angle of attack, and loads on the model were measured and an estimate of the error of each measurement was made.

From an error propagation point of view, the measurement errors associated with the operating conditions can be propagated through the analysis tool along with an estimate of the error introduced by the analysis tool itself to verify that the analysis agrees with the

experiment. If the error bounds propagated through the analysis overlap with the error bounds of the force balance measurement, the analysis agrees with the experiment. If the error in the prediction were found to be too great, the impact of each source of error can be investigated to suggest a course of action to reduce the overall error. This could include improved measuring equipment or an improvement to the analysis tool. Once agreement is achieved, a parametric study of the analysis can be performed to predict the model's behavior with an understanding of the error associated with the predictions.

From a robust design point of view, the experimenter may wish to confirm that the model loads will not exceed a certain level for a certain proportion of the operating life of the model i.e. the loads must less than 200 lbf for 85% of the expected 1000 hr experiment life cycle. In this situation, the analyst must model the operating life of the experiment, not just a single given day. Local weather data can yield statistical distributions for atmospheric temperature and density during the tunnel operating hours in the season that the experiments will be conducted. The analyst may assume that each angle of attack within the permissible range is equally likely to be investigated. Uncertainty propagation techniques may be used to propagate the uncertain operating conditions of the experiment through the analysis tool in order to confirm the robustness criterion is met.

In this robust design example, concepts such as the error in angle of attack or tunnel velocity measurement have no meaning. Similarly, in this error propagation example, concepts such as the seasonal variation of ambient operating conditions have no meaning. In robust design, quantities are not known exactly, but are drawn from an uncertain distribution of equally valid values. In error allocation, quantities have one true value, but this value is not perfectly known.

Of course, situations arise where it is sometimes difficult to classify a change as uncertainty or error. The final verdict usually depends on the perspective of the decision maker, and the goals of the particular study. For instance, the angle of attack of the model in the example may be dependent on the aerodynamic forces experienced by the model; this effect may even be unsteady in nature. Whether this change is best treated as uncertainty or a source of error depends on the perspective of the decision maker.

4.3 Types of Error

Orthogonal to the distinction between error and uncertainty, error and uncertainty can be classified as being systematic (epistemic) or random (aleatory) [23]. Systematic error is that which produces a bias, an example is an analysis code that is known to underpredict a certain result by 5%. Random error is that which produces scatter about the correct value, an example is an analysis code that is known to be correct within 5%.

Systematic error can be considered determinate error and can be propagated as discussed later in this chapter. However, if a systematic error source is known, the bias may be eliminated by applying an appropriate correction to the analysis. Systematic error is not subject to the statistical analysis techniques used in this thesis. This thesis assumes any source of systematic bias has been corrected, leaving only random error.

Some researchers introduce an additional class of error. Unacknowledged error includes blunders or mistakes in the analysis process. Unfortunately, there are no systematic methods for dealing with unacknowledged errors [23], so one must assume the unacknowledged errors are not present.

Another classification of error can be made, errors can be uniform, or they can vary throughout the design space (called mapped errors). In actuality, almost all error sources are mapped. However, in most situations, the variation of error throughout the design space will not be well understood. Furthermore, asking the user to define an error map would be too cumbersome for an intuitive user interface. Consequently, most of the errors discussed in this document are treated as uniform sources of error. This simplification was made to improve the accessability and usability of the tool, not in reaction to any limitation of the theory and methods used herein; the error propagation technique presented in this work applies directly to mapped error sources.

4.4 Quantifying Error

The critical tasks of identifying and quantifying error must be completed before the impact of error may be studied or understood. While the identification and quantification of error are sometimes considered as separate activities, they are in fact the same activity.

Put another way, there is little need to identify an error source because nearly everything introduces error. Most error identification activities are actually concerned with identifying *significant* sources of error. This qualifier reveals that the central activity is actually the quantification of errors with the option of discarding those that fall beneath some certain threshold.

In this context, the identification of error is really an act of partitioning the total error into quantified parts attributed to various error sources.

Unfortunately, the process of validating a tool, thereby quantifying its error, is itself quite challenging. Error is quantified by comparing a tool with some sort of benchmark. Ideally, the benchmark is an observance of a system in real world conditions. Frequently, the benchmark is a highly controlled laboratory experiment. Sometimes, the benchmark is a well trusted analysis tool.

Validation studies should be performed for a number of cases similar to those of interest. This repetition is needed to build a statistically significant sample. Validation studies must also be repeated whenever the problem of interest changes. The validation of a tool for one problem may not imply anything about the validity of the tool for another problem.

This research does not address the challenges inherent to tool validation. Instead, it is assumed that the error has been quantified by an appropriate validation study in accordance with good practice. No tool should be applied without an understanding of the error introduced by applying the tool to a particular problem.

4.5 Reducing Error

The concept of a fidelity trade is made relevant by the ability to reduce a chosen source of error. While the presence of error is a universal truth, error can be reduced. Of course, there is no universal means to reduce any error; error reduction techniques depend on the error source and on how large an improvement is needed. Recall that this discussion pertains to random error. Any systematic error that is present, known, and quantified should be corrected out of the tool.

Reducing an error is directly tied to the identification and quantification of the error

source. For example, in a CFD or FEA study, once a grid resolution error has been identified and quantified, grid resolution improvements can reduce the error. An error identified as being due to the physical simplifications inherent in an analysis tool can be reduced by removing or improving some of those physical simplifications.

Every error source and any means to improve fidelity should be considered when examining the system. Operational changes to existing analysis codes (convergence tolerance, grid resolution, solution adaptation, etc.) are easy to achieve. However, tool modification, switching tools, or replacing a tool with experimentation can have a large impact on error.

4.6 Error Propagation

In addition to error inherent to a calculation, in any calculation the error in any input quantity contributes to an erroneous output. The magnitude of error in an output may be diminished or amplified relative to the error of the inputs. In a series of calculations representing a complex system, error can build up and interact in non-intuitive ways. The behavior of error becomes even more obscure when the series of calculations are coupled.

Error propagation is a set of techniques used to quantify and understand the error in an output quantity based on some knowledge of the error of the input quantities and error introduced during the process. Error propagation was originally developed for the experimental sciences [24, 25]. In an experiment, the source of all error is measurement, as Mother Nature makes no mistakes. Conversely, computer analysis codes have many sources of error. They are subject to error in their inputs and their process but there is no measurement error in their outputs [26, 23].

Error propagates through every component of a complex system. Understanding the error propagation through a complex system starts with understanding error propagation through an individual component. The JMP statistics and data analysis software recently added the capability to propagate error through individual components using techniques similar to those used in this research [27]. Of course, few complex systems can be adequately modeled with a single black box. Because of the interaction and flow of information through a complex system, the error propagation through a complex system is necessarily more

complicated.

Error is introduced at every point in a complex system. Every error source propagates through the complex system in the same way information propagates. If a system has feedforward, the errors feed forward. If a system has feedback, then errors feed back. If a system has coupled loops, then errors are coupled.

As with error propagation through a single analysis, error sources grow and decay when propagated through a complex system. These behaviors are made more complex by the complex interactions of a complete system. This complication and the loss of intuition that accompanies it makes proper error propagation calculations even more important to aid the decision maker.

The complex system is modeled as a system of equations that must be iterated to achieve compatibility. This iteration greatly increases the expense of exploring a design space. Re-converging a complex system in this manner to investigate the impact of a potential error source would be very expensive. Instead, error propagation is combined with system sensitivity analysis and matrix inversion to instantly propagate an error through the complex system without re-converging the system.

4.6.1 Nomenclature

Error propagation finds its roots in both calculus and statistics leading to a notation which may not be immediately familiar. Error propagation deals with the subtly related quantities of a differential, derivative, deviation, uncertainty, and variance. Each of these quantities is discussed in a local and propagated sense—partial and total quantities. Informal definitions of these quantities including the symbols used to represent the quantities follow. The switch from operator to subscript notation is done to match differing conventions in the calculus and statistics communities.

Derivative The partial derivative of a quantity, f, with respect to another quantity, x, may be written $\frac{\partial f}{\partial x}$. In the conventional manner, this represents the infinitesimal change in f due to a unit infinitesimal change in x while all other quantities are held constant. If one allows other quantities to vary and adjust with the change in x, one arrives at the total

derivative, $\frac{df}{dx}$. The distinction between the partial and total derivative is analogous to the distinction between a local and propagated error, where the total (propagated) quantity reflects the entire ripple-effect of a change in x. This distinction applies to the rest of the definitions included herein.

Differential The partial differential of a quantity may be written ∂f. The differential of a quantity is the infinitesimal change normalized via division by a differential in another quantity to obtain the derivative. The total differential is written df.

Deviation The partial deviation of a quantity, written δf, is a known finite local change in the quantity. The total deviation is written Δf.

Uncertainty The partial uncertainty of a quantity, written ϵ_f, is a random variable of finite change in a quantity. The total uncertainty is written ε_f. The uncertainty is usually assumed have zero mean. The use of the term uncertainty in this context is separate from the distinction made earlier in this chapter. In this context, uncertainty merely refers to a change in a quantity which can be described by a random variable.

Variance The variance of the partial uncertainty of a quantity is written ς_f; the variance of the total uncertainty is written σ_f.

4.6.1.1 System Description

A coupled system of systems, no matter how complex, may be described as a functional black box, $\mathbf{f}(\mathbf{x})$, where $\mathbf{f}$ is a vector of m functions of $\mathbf{x}$, a vector of n variables.

The system, $\mathbf{f}$, is broken into l subsystems, $\mathbf{f} = \mathbf{g}_i;\ i = 1, \ldots, l$, each a vector of m_i functions $\mathbf{g}_i = g_{i,j};\ j = 1, \ldots, m_l$ of $\mathbf{x}$ and the other subsystems; this is written $\mathbf{g}_i(\mathbf{x}, \mathbf{g}_{k \neq i})$. Here, the comma notation $g_{i,j}$ is used to mean the jth element of vector $\mathbf{g}_i$, it does not imply that $\mathbf{g}$ is a matrix, nor does it imply differentiation with respect to j. The decomposition

of the system into subsystems is further illustrated by the following equation.

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} \equiv \begin{bmatrix} \mathbf{g}_1(\mathbf{x}, \mathbf{g}_{k\neq 1}) \\ \mathbf{g}_i(\mathbf{x}, \mathbf{g}_{k\neq i}) \\ \vdots \\ \mathbf{g}_l(\mathbf{x}, \mathbf{g}_{k\neq l}) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} g_{1,1}(\mathbf{x}, \mathbf{g}_{k\neq 1}) \\ g_{1,j}(\mathbf{x}, \mathbf{g}_{k\neq 1}) \\ \vdots \\ g_{1,m_1}(\mathbf{x}, \mathbf{g}_{k\neq 1}) \end{bmatrix} \\ \begin{bmatrix} g_{i,1}(\mathbf{x}, \mathbf{g}_{k\neq i}) \\ g_{i,j}(\mathbf{x}, \mathbf{g}_{k\neq i}) \\ \vdots \\ g_{i,m_i}(\mathbf{x}, \mathbf{g}_{k\neq i}) \end{bmatrix} \\ \vdots \\ \begin{bmatrix} g_{l,1}(\mathbf{x}, \mathbf{g}_{k\neq l}) \\ g_{l,j}(\mathbf{x}, \mathbf{g}_{k\neq l}) \\ \vdots \\ g_{l,m_l}(\mathbf{x}, \mathbf{g}_{k\neq l}) \end{bmatrix} \end{bmatrix}$$

The total number of functions in the system, m is equal to the sum of the number of functions in each of the subsystems m_i. i.e. $m = \sum_{i=1}^{l} m_i$

4.6.2 Total Differential

Writing out the definition of the total differential of the subfunctions yields Equation 1. This equation intuitively represents the influence of the coupled components on one another. If one quantity were to be infinitesimally perturbed in some way, Equation 1 predicts the influence of that perturbation on the entire system. The first term on the right hand side of Equation 1 accounts for the impact of a perturbation to a quantity on that quantity while the second term accounts for the impact of the perturbation to the quantity due to the interactions with all the other components. Unfortunately, this equation is difficult to apply because total differential terms appear on both sides of the equation. This equation says nothing about the cause or source of the perturbation.

$$d\mathbf{g}_i = \partial \mathbf{g}_i + \sum_{j=1, j\neq i}^{l} \frac{\partial \mathbf{g}_i}{\partial \mathbf{g}_j} d\mathbf{g}_j \tag{1}$$

The differential terms $d\mathbf{g}_i$ and $\partial\mathbf{g}_i$ are both $m_i \times 1$ vectors. Whereas the sensitivity term, $\frac{\partial \mathbf{g}_i}{\partial \mathbf{g}_j}$, is a $m_i \times m_j$ vector.

4.6.2.1 Application to Derivatives

If the total differential is normalized by differentials of another variable, $\mathbf{x}$, Equation 2 for the total derivative with respect to $\mathbf{x}$ results. The total derivative predicts the normalized influence of a perturbation of an input throughout the system.

$$\frac{d\mathbf{g}_i}{d\mathbf{x}} = \frac{\partial \mathbf{g}_i}{\partial \mathbf{x}} + \sum_{j=1, j\neq i}^{l} \frac{\partial \mathbf{g}_i}{\partial \mathbf{g}_j} \frac{d\mathbf{g}_j}{d\mathbf{x}} \tag{2}$$

Where $\mathbf{x}$ is a $n \times 1$ vector, and the remaining terms are matrices, where $\frac{\partial \mathbf{g_i}}{\partial \mathbf{x}}$ and $\frac{d \mathbf{g_i}}{d \mathbf{x}}$ are both $m_i \times n$ and $\frac{\partial \mathbf{g}_i}{\partial \mathbf{g}_j}$ is $m_i \times m_j$.

When Equation 2 is multiplied by differentials of $\mathbf{x}$, the normalized perturbations represented by the derivatives are scaled according to arbitrary perturbations in $\mathbf{x}$, as shown in Equation 3.

$$d\mathbf{g}_i = \frac{\partial \mathbf{g}_i}{\partial \mathbf{x}} d\mathbf{x} + \sum_{j=1, j\neq i}^{l} \frac{\partial \mathbf{g}_i}{\partial \mathbf{g}_j} \frac{d\mathbf{g}_j}{d\mathbf{x}} d\mathbf{x} \tag{3}$$

Here, the left-hand side of the equation has been simplified to reflect its status as a total differential. The meaning of Equation 3 is analogous to Equation 1 except that the source of the perturbation is no longer entirely arbitrary. Equation 3 predicts the impact of arbitrary perturbations in $\mathbf{x}$ on the coupled system. Unfortunately, this equation is difficult to apply because the $\frac{d\mathbf{g}_j}{d\mathbf{x}}$ term is difficult to obtain.

4.6.2.2 Application to Finite Increments

The total differential is a statement of how a coupled system will respond due to infinitesimal changes in the system. The finite increment case analogous to Equation 1 is shown below and amounts to a linearized form of the differential which is exact for small increments.

$$\Delta\mathbf{g}_i \approx \delta\mathbf{g}_i + \sum_{j=1, j\neq i}^{l} \frac{\delta \mathbf{g}_i}{\delta \mathbf{g}_j} \Delta\mathbf{g}_j$$

As before, the increments applied in this case are not necessarily related through any common source; they may be arbitrarily prescribed. The finite form of Equation 3, given

below, predicts the impact caused by arbitrary finite increments in another variable on the system.

$$\Delta \mathbf{g}_i \approx \frac{\partial \mathbf{g}_i}{\partial \mathbf{x}} \Delta \mathbf{x} + \sum_{j=1, j \neq i}^{l} \frac{\partial \mathbf{g}_i}{\partial \mathbf{g}_j} \frac{d\mathbf{g}_j}{d\mathbf{x}} \Delta \mathbf{x}$$

Of course, these finite increment forms of the differential equation are difficult to apply because of the total differential term appearing on the right-hand side of the equation.

4.6.3 System Sensitivity Analysis

System sensitivity analysis (SSA) [28] is a technique developed for multi-disciplenary optimization (MDO) which finds equal applicability to the error propagation of complex systems. SSA addresses the difficulty outlined in the previous section, that the total differential of a component depends on the total differentials of all the other components in the system. Equation 1 may be rearranged to the following form.

$$\partial \mathbf{g}_i = d\mathbf{g}_i - \sum_{j=1, j \neq i}^{l} \frac{\partial \mathbf{g}_i}{\partial \mathbf{g}_j} d\mathbf{g}_j$$

Using the definitions presented in Section 4.6.1.1, this equation may be written in matrix form as follows.

$$\partial \mathbf{f} = \begin{bmatrix} \mathbf{I}_1 & -\frac{\partial \mathbf{g}_1}{\partial \mathbf{g}_2} & \cdots & -\frac{\partial \mathbf{g}_1}{\partial \mathbf{g}_l} \\ -\frac{\partial \mathbf{g}_2}{\partial \mathbf{g}_1} & \mathbf{I}_2 & & \vdots \\ \vdots & & \ddots & -\frac{\partial \mathbf{g}_{l-1}}{\partial \mathbf{g}_l} \\ -\frac{\partial \mathbf{g}_l}{\partial \mathbf{g}_1} & \cdots & -\frac{\partial \mathbf{g}_l}{\partial \mathbf{g}_{l-1}} & \mathbf{I}_l \end{bmatrix} d\mathbf{f}$$

The matrix in the above equation is the system sensitivity matrix, $\mathbf{M}$, which is $m \times m$. As expected, $\partial \mathbf{f}$ and $d\mathbf{f}$ are both $m \times 1$ and $\mathbf{I}_i$ is an $m_i \times m_i$ identity matrix. Using this definition, the equation may be compactly written as shown below.

$$\partial \mathbf{f} = \mathbf{M}\, d\mathbf{f}$$

Inversion of the system sensitivity matrix isolates all total differential terms on the left hand side of the equation as shown below in Equation 4.

$$d\mathbf{f} = \mathbf{M}^{-1} \partial \mathbf{f} \tag{4}$$

This is an equation for the total differential of a coupled system of equations in terms of only partial differentials. This important result addresses the difficulty encountered in the previous section, significantly improving the ease, accuracy, and cost of the calculation of a total differential.

The system sensitivity matrix $\mathbf{M}$ completely describes component interactions. Upper-triangular terms represent feedback while lower-triangular terms represent feedforward. Zero terms indicate independence while large terms indicate strong dependence. Visualization of the sensitivity matrix yields a result substantially similar to a Design Structure Matrix (DSM) diagram [28, 10].

4.6.3.1 Application to Derivatives

The process undertaken for the derivation of the total derivative (Equation 1 to Equation 2) applies analogously to Equation 4 to arrive at Equation 5 for the total derivative entirely in terms of partial derivatives.

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{x}} \tag{5}$$

4.6.3.2 Application to Finite Increments

Equation 4 may be trivially extended to arrive at its finite increment form given below as Equation 6. This is a formula revealing how arbitrary finite increments in an output interact and propagate through a coupled system thus impacting all the outputs.

$$\Delta\mathbf{f} \approx \mathbf{M}^{-1}\delta\mathbf{f} \tag{6}$$

Analogously, the impact of finite increments in $\mathbf{x}$ may be applied to Equation 5, resulting in Equation 7 given below.

$$\Delta\mathbf{f} \approx \mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Delta\mathbf{x} \tag{7}$$

The equations for the finite increment in the system due to finite increments in the components or input variables are linearizations of the exact equation; they are exact for small increments. These equations could also be derived by taking the first term of the Taylor series expansion for the functions; as such, they are first-order accurate.

Equations 6 and 7 may be used to model the behavior of the system when one of the components or inputs (respectively) are changed. For example, if a particular component output is known to be 5% too small, Equation 6 will predict the impact of the error on the system. This foreshadows the way error propagation will be modeled through a system, but this theory is not complete.

As expected, the finite increment in a system subject to finite increments in components and input values is simply the sum of the individual increments. The terms may be grouped and the equation may be written as Equation 8.

$$\Delta\mathbf{f} \approx \mathbf{M}^{-1}\left(\delta\mathbf{f} + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\Delta\mathbf{x}\right) \tag{8}$$

4.6.4 Determinate Error

In error propagation, $\bar{u}$ is interpreted as the nominal value of an observation u which is subject to error. The absolute error in an observation u is given by $\epsilon_u = u - \bar{u}$. Thereby, u and ϵ_u may be treated as random variables centered at $\bar{u}$ and zero, respectively.

If the error in a set of inputs and component values were somehow known they could be inserted into Equation 8 as finite increments and the error in the system could be evaluated as shown below. This is called determinate error. Unfortunately, error is seldom determinate. Determinate error is another name for systemic (epistemic) error.

$$\varepsilon_{\mathbf{f}} \approx \mathbf{M}^{-1}\left(\epsilon_{\mathbf{f}} + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\varepsilon_{\mathbf{x}}\right)$$

4.6.5 Bounded Error

Usually, less is known about error. Frequently, only error bounds or tolerances are available; they are written in a form such as $\bar{u} \pm \epsilon_u$. This form implies knowledge about the maximum magnitude of an error, but no knowledge of the magnitude or sign of a particular error.

The worst-case uncertainty is calculated by assuming all contributing errors are at the positive upper bound and applying those quantities exactly as with determinate error. In this case, the sign of the error has been left undetermined, while a worst-case assumption has been made to determine the magnitude of the error. This results in an accurate error

bound for the system. Unfortunately, this result is usually too conservative to be of practical use.

4.6.6 Indeterminate Error

Instead of making a worst-case assumption about the behavior of error, it may be treated comprehensively as a random variable. The variance of N observations of u with the mean $\bar{u}$ is defined as follows.

$$\sigma_u^2 \equiv \frac{1}{N-1}\sum_{l=1}^{N}(u_l - \bar{u})^2 \tag{9}$$

The covariance of N observations of u and v with means $\bar{u}$ and $\bar{v}$ is defined as follows.

$$\sigma_{uv}^2 \equiv \frac{1}{N-1}\sum_{l=1}^{N}\left[(u_l - \bar{u})\,(v_l - \bar{v})\right]$$

Recognizing the $u_l - \bar{u}$ term as the l^{th} observation of the random variable ϵ_u, and applying the variance definition to the entire system yields the following equation.

$$\sigma_{\mathbf{f}}^2 = \frac{1}{N-1}\sum_{l=1}^{N}(\Delta\mathbf{f})^2 \tag{10}$$

Here, and henceforth, the l subscript and its limits associated with the summation have been dropped for clarity. The use of Equation 9 as a vector equation is noteworthy: the square operation is carried out in a term-by-term sense, not in a vector sense. Substitution of Equation 6 into this equation yields the following.

$$\sigma_{\mathbf{f}}^2 \approx \frac{1}{N-1}\sum\left(\mathbf{M}^{-1}\delta\mathbf{f}\right)^2$$

This expression is approximate only in the sense that Equation 6 is a linearized approximation of the exact impact of finite increments on the system. Expansion of the square term deserves special attention due to its term-by-term nature. This term may be written in subscript notation as follows.

$$(\Delta f_i)^2 \approx \left[\left(M^{-1}\right)_{ij}\delta f_j\right]^2$$

Alternatively, this may be written as follows by expanding the square.

$$(\Delta f_i)^2 \approx \left[\left(M^{-1}\right)_{ij}\delta f_j\left(M^{-1}\right)_{ik}\delta f_k\right]$$

Due to the term-by-term nature of the square operation, the repeated i subscript is *not* summed across, while the repeated j and k indices are. The terms of the summation fall into two major categories: those where j and k are equal and those where they are not; respectively these are the first and second term of the right hand side of the following equation.

$$(\Delta f_i)^2 \approx \left[\left(M^{-1}\right)_{ij}\right]^2 (\delta f_j)^2 + \left[\left(M^{-1}\right)_{ij} \delta f_j \left(M^{-1}\right)_{ik} \delta f_k\right]_{j \neq k}$$

All the square operations are performed on a term-by-term basis. Substitution of this expression into Equation 10 yields.

$$\sigma_{\mathbf{f}}^2 \approx \frac{1}{N-1} \sum \left\{ \left[\left(M^{-1}\right)_{ij}\right]^2 (\delta f_j)^2 + \left[\left(M^{-1}\right)_{ij} \delta f_j \left(M^{-1}\right)_{ik} \delta f_k\right]_{j \neq k} \right\}$$

The summation may be split, and because only the f term varies with the implied summation index, the system sensitivity matrix inverse may be pulled outside the summation.

$$\sigma_{\mathbf{f}}^2 \approx \left[\left(M^{-1}\right)_{ij}\right]^2 \frac{1}{N-1} \sum (\delta f_j)^2 + \left[\left(M^{-1}\right)_{ij} \left(M^{-1}\right)_{ik} \frac{1}{N-1} \sum (\delta f_j \delta f_k)\right]_{j \neq k}$$

The scaled summations are recognized as the partial (unpropagated) variance and covariance of f_j and $f_j f_k$, respectively.

$$\sigma_{\mathbf{f}}^2 \approx \left(\mathbf{M}^{-1}\right)^2 \varsigma_{\mathbf{f}}^2 + \left[\left(M^{-1}\right)_{ij} \left(M^{-1}\right)_{ik} \varsigma_{f_j f_k}^2\right]_{j \neq k}$$

When δf_j and δf_k are uncorrelated, their covariance is identically zero ($\varsigma_{f_j f_k} \equiv 0$). Any two unpropagated errors will be independent random variables and therefore uncorrelated. Therefore, in the context of error propagation, the second term may be dropped; this results in Equation 11 for the propagation of indeterminate error. Recall that the square operation on the vector and matrix terms is done on a term-by-term basis.

$$\sigma_{\mathbf{f}}^2 \approx \left(\mathbf{M}^{-1}\right)^2 \varsigma_{\mathbf{f}}^2 \tag{11}$$

The derivation of Equation 11 demonstrates a Pythagorean sum as a technique for combining independent random variables. In the same way, a Pythagorean sum may be used to combine any independent sources of indeterminate error. Straightforward application of

a Pythagorean sum to Equation 8 results in the following equation for the propagated indeterminate error of a system subject to error in components and inputs.

$$\sigma_{\mathbf{f}}^2 \approx \left(\mathbf{M}^{-1}\right)^2 \left[\varsigma_{\mathbf{f}}^2 + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)^2 \sigma_{\mathbf{x}}^2\right] \tag{12}$$

This equation may be used to track the propagation of error throughout an arbitrarily complex system. Multiple sources of component error, say physical fidelity, numerical convergence, and metamodel approximation, may all be handled through an appropriate Pythagorean sum. The same approach can be used for multiple sources of input error.

The assumption of the independence of separate error sources deserves further comment. It is essentially a statement that when one source of error is positive and large, nothing can be said about the magnitude or sign of another error source. For example, if a drag estimation significantly overestimates the drag on a vehicle, nothing conclusive can be said about the error in a weight estimation of the same vehicle.

Monte Carlo analyses were performed to verify the sensitivity approach to error propagation for each of the example systems presented in this thesis. The results of these studies were included with the discussion of each system.

4.7 Error Stability

Equation 12 can be used to propagate an arbitrary combination of sources of error through an arbitrarily complex system. The behavior of error in a complex system can sometimes be counterintuitive. In order to understand this behavior, it is instructive to examine some simplified situations in the context of Equation 12.

First, consider the situation where there is no error in any of the input quantities. In this situation, the $\sigma_{\mathbf{x}}^2$ vector is a zero vector, and Equation 12 simplifies to Equation 11.

Further, consider the situation where only one error source is introduced at any time. In this situation, the squared vector of error sources, $\varsigma_{\mathbf{f}}^2$, has only one nonzero term.

In this situation, the matrix-vector multiplication simplifies to a vector-scalar multiplication, where the vector corresponds to the appropriate row of the matrix, and the scalar corresponds to the nonzero term of the error source vector. Of course, the vector-scalar

multiplication does not have the dot-product-like implied summation of terms present in a matrix-vector multiplication. Furthermore, without the implied summation, we can safely take the square root of both sides of the equation.

Therefore, in the situation of a solitary source of error, the propagation of error loses the Pythagorean sum. This situation matches intuition; after all, when one side of a right triangle is of vanishingly small length, the other side's length is equal to the length of the hypotenuse.

Furthermore, inspection of the diagonal of the $\mathbf{M}^{-1}$ matrix yields insight into the stability of an error source in the complex system. When an individual source of error is introduced to a quantity in a system, the output error level of that quantity is determined by the corresponding element of the diagonal of the $\mathbf{M}^{-1}$ matrix. If the corresponding term of the diagonal is greater than one, the error in the system output will be greater than the error introduced. However, if the corresponding term of the diagonal is less than one, the error in the system output will be less than the error introduced. The term error stability is used in an analogy to aircraft stability; a stable error will tend to be damped by the system while an unstable error will tend to be amplified by the system.

The counterintuitive decay of an error source in a complex system can be understood in terms of system stability. For some quantities, the system tends to amplify an error source; whereas, for other quantities, the system tends to diminish an error source. Whether any particular error source grows or decays is dependent on the overall system behavior at that point in the design space.

Whether a source of error grows or decays is determined by its diagonal of the $\mathbf{M}^{-1}$ matrix. Of course, the diagonal elements of the $\mathbf{M}$ matrix are all equal to one. Consequently, whether the diagonal terms in the $\mathbf{M}^{-1}$ matrix are greater or less than one is determined by the off diagonal terms of the system sensitivity matrix ($\mathbf{M}$). A numerical example of this phenomenon is examined in Appendix A.

4.8 Error Attribution

As demonstrated in the previous sections, a statistical treatment of error results in error sources combining through a Pythagorean sum. It is useful to be able to quantify how each source contributes to the propagated error. This allows the decision maker identify an individual error source for improvement. Because of the Pythagorean sum, individual errors are not additive in a linear sense. It is then difficult to portion the total error into segments due to each of the contributors. In fact, in light of the Pythagorean sum, it is not obvious exactly what such a division represents. However, the obvious utility of such a division makes some amount of ambiguity worthwhile.

In order to divide the total error into parts due to each contributor, the propagated error due to each source is first calculated in isolation. The total error is the Pythagorean sum of these terms, the numerator in the below equation. The algebraic sum of these error terms is significantly larger than the Pythagorean sum, the denominator in Equation 13. Recall that the lengths of the sides of a right triangle always sum to be greater than the length of the hypotenuse. Next, each contributor is scaled by the ratio of the Pythagorean sum to the algebraic sum. This procedure resulted in a division of error with generally intuitive behavior.

$$\bar{\sigma}_j = \frac{\sqrt{\sum \sigma^2}}{\sum \sigma} \sigma_j \tag{13}$$

4.9 Error Allocation and Fidelity Trades

Error has been shown to be ubiquitous. Every step of the systems design process can introduce a source of error. Error sources may be reduced but not eliminated. The cost to achieve a given reduction in a given error varies widely. Differing error sources contribute differently to the total system error. Some errors grow while others decay. The propagation of error through a complex system is nonintuitive, but it can be calculated. The contribution of each error source to the total system error can be quantified.

These traits describing error in a complex system imply that error can be comprehensively considered by the decision maker. For example, a final error goal can be set, and the

contributions to the error can be budgeted (allocated) among the contributors. Similarly, improvements to the fidelity of the various contributors can be weighed (traded) based on their relative cost and benefit to the system. The designer can understand the impact of error on his choice of a design, as well on the predicted performance of the design.

4.10 Error Management Interface

Visualization provides a unique opportunity to provide insight into the design process and interactive dynamic visualization further enhances that insight. Various visualization techniques have been developed to aid in complex systems design. These techniques typically strive to make tractable the vast interacting multivariate network of components that make up a complex system. Interactive visualization tools can also facilitate methods of work which allow the user to investigate, understand, and guide the design process in new and powerful ways. The visualization GUI, with interactive error analysis tools, becomes an error management interface for performing a new kind of tradeoff.

This research will work to extend existing dynamic visualization techniques to capitalize on the additional information provided on error, its sources and propagation. Furthermore, an error management interface will be developed to provide intuitive interaction with error cause and effect. There will be an interactive prediction profiler capable of displaying dynamic confidence intervals around the responses, contour plots with dynamic constraints modified to incorporate dynamic error bounds, and a system integration visualization depicting the flow of information through the system. As stated previously, these representations of error and its propagation will be implemented in a way that does not detract from the clarity of conventional design space visualization.

The error management interface will combine these design space depictions with a set of intuitive fidelity controls. The user will be able to set a target level of fidelity or an assumption on the fidelity of a particular analysis code output or system level input; these error levels will be propagated throughout the complex system. Setting a target level of fidelity and back-propagating that error implies an error budget by which error may be allocated to each discipline and quantity throughout the system. A quantified error budget

and allocation interface will allow the user to identify "heavy hitters" and "low hanging fruit," in terms of system fidelity, thus providing guidance for the appropriate use of—and investment in—higher fidelity analysis tools.

In order to comprehensively consider the impact of error, the decision maker must have access to an appropriate tool. Good decision making tools provide intuitive access to the information needed to make decisions and mechanisms to act on those decisions.

As a field, complex systems design has already developed many needed decision making tools. To bring error into consideration, it is best to start with the existing tools and augment them with information access and action mechanisms related to error; additional tools can be created as required.

Starting with pre-existing tools is of clear benefit, leveraging the decision maker's experience to speed familiarity with the new decision making domain. The user can immediately see the impact of the new information on familiar terms.

Existing information displays can be modified to intuitively display the impact of error through the display of error bounds surrounding the quantities of interest. These bounds will usually correspond to the total propagated error. In addition to existing information displays, the consideration of various error sources suggests that a new display is needed to depict how each individual error source contributes to the total. This display corresponds directly to error attribution as discussed above.

The action mechanisms related to error include the ability to add, remove, and modify sources of error associated with any quantity relevant to the complex system. These actions should be completed rapidly, providing interactive feedback for the decision maker.

At present, many systems design visualization tools present disjointed views of the system model. Because the views are not interconnected, when the designer changes from one view to another, he must transfer any variable settings or decisions to the new view. If he makes some progress, but wishes to switch back to the original view, he must once again transfer the information manually. This disconnect presents a significant barrier to switching design views freely. Instead, the designer should have the freedom to choose which system view is most important at every stage of the decision making process. This freedom

becomes more important as the decision making domain available to the designer grows.

In addition to allowing linked views to facilitate the decision maker's preferred workflow, the design interface should present a dynamically reconfigurable interface made of the various views and controls. With a dynamically reconfigurable interface, the designer can focus on the views and controls most significant for each phase of the decision making process without superfluous clutter.

CHAPTER V

METAMODELING

Metamodels use a set of sample data to build an approximate model of the function used to evaluate the sample data. The metamodel may then be used as a surrogate for the original function; this enables entirely new approaches to design and allows design studies to be carried out easier, faster, and cheaper. There are a wide variety of metamodeling techniques available for use, each with its own characteristics, strengths, and weaknesses.

Using metamodels as a surrogate to an analysis is an enabling technique for many new approaches to design including large scale optimization, improved visualization, and probabilistic design. In this research, metamodeling provides an enabling technique for the propagation of error through a complex system and the visualization of the behavior of the complex system while considering error.

5.1 Local vs. Global Metamodels

Metamodels may be sorted into two categories which generalize many of these characteristics: those with global behavior and those with local behavior. For a thorough survey of metamodeling techniques in the context of design, please see Simpson et al. [29] and Jin et al. [30].

5.1.1 Global Behavior

Global metamodels are those where a change in the value of one sample point (or the addition of a point) changes the metamodel value globally, i.e. throughout the entire domain.

Global metamodels typically assume a global form of the function. Parameters are then adjusted such that the metamodel best matches the sample data. Most least squares regression techniques, including polynomial response surfaces, are global metamodels.

5.1.2 Local Behavior

Local metamodels are those where a change in the value of one sample point (or the addition of a point) changes the metamodel value locally, i.e. in the neighborhood of the change.

Local metamodels typically assume a local form of the function. Parameters are then adjusted such that the metamodel best matches the sample data. Moving least squares regression and any sort of interpolation, including linear interpolation or a cubic spline, are local metamodels.

5.1.3 Form of Model

Global metamodels assume a global form of the function being modeled. This can be very powerful when something about the underlying behavior of the system being modeled is known, especially when the sample data is noisy.

When nothing about the underlying behavior of the system is known, it is inappropriate to make assumptions about the behavior of the function. In practice, this pitfall is avoided by selecting a domain for the metamodel small enough that any behavior unlike the assumed behavior is minimized; after all, everything is linear given a small enough scale. For large, adaptive ranges, this will not be the case. However, when an assumed form is used, and satisfactory metamodels are obtained, examination of the resulting equation can yield insight into the underlying behavior.

Transformations of the inputs or outputs to a metamodel may be used to adjust the assumed behavior of the metamodel to match the behavior of the function. This is best done when something about the form of the function is known, but may also be done by trial and error to find a best set of transformations [31, 32]. Transformations can also improve a local metamodel but, by their nature, the trial and error method of choosing a transformation will not work with a local metamodel.

Local metamodels, on the other hand, assume at most a local form of the function being modeled. Complex global behavior is built from the combination of all the simple local behaviors. If some aspect of a function's behavior is not being captured, additional sample points in the region will improve the metamodel. This approach leverages the fact

that complex behavior is simple on a small scale without sacrificing performance over large scales.

5.1.4 Analytical Form

Global metamodels can often be expressed in a simple analytical form. They also typically are defined by a finite number of parameters. Local metamodels usually have complex implementations without a simple analytical form. The defining parameters are frequently unavailable, and often all sample points must be maintained when transmitting the metamodel.

These characteristics mean that global metamodels are frequently easier to transfer and implement. Also, it is often possible to obtain analytical derivatives of global metamodels. This is useful for complex problems including optimization. As discussed in Chapter 4, the use of sensitivity techniques for error propagation require the efficient and accurate prediction of derivatives. Local metamodels sometimes have easily obtainable analytical derivatives.

5.1.5 Noisy vs. Deterministic Data

In the case of this research, the sample data is deterministic; the metamodel is used to approximate computer codes guaranteed to return the same response given the same inputs [33]. Whereas, if one was attempting to fit a metamodel to experimental observations, one would expect some amount of random noise in the measurement; repeated experiments are not guaranteed to return the same results.

With a global metamodel, the model usually has fewer degrees of freedom (parameters to adjust) than there are sample points in the data set. In this case, unless the assumed form is exactly correct and the sample points are noise-free, it is impossible for the resultant model to pass *through* all the sample points. Instead, the model passes as close to all of the points as possible. In situations where the sample data is noisy (it has random error) this smoothing property is desirable; this is especially true when something about the underlying functional form is known.

With a local metamodel, the model usually passes directly through all the points in the

data set. In situations where the sample data is deterministic this property is desirable. Some local metamodels can be used to model noisy data.

5.1.6 Error Estimates

The evaluation of the quality of a metamodel is central to its use as a design tool. This brief discussion will be restricted to error estimation in a local vs. global metamodeling context.

Not surprisingly, the built-in error estimates that accompany global metamodels are usually global in nature while those that accompany local metamodels are usually local in nature. Global quality metrics are not as useful for local adaptation as are local metrics. Global quality metrics are often most appropriate for use with noisy data sets rather than the deterministic data encountered in this research. Conversely, local metamodels may have built-in error estimates that are local in nature.

Techniques for estimating the error of a metamodel may be classified as analytical or empirical. Analytical approaches are able to estimate the error of a metamodel without executing a test case or manipulating the sample data in any way. Empirical approaches estimate the error by either running additional test cases or by removing points from the sample data set.

If an empirical approach to evaluating the quality of a metamodel must be used, another important difference between local and global metamodels arises. The test points used to investigate a local metamodel may then be used to improve the local metamodel; the test points used to investigate a global metamodel must be thrown away and wasted.

5.1.7 Training

All metamodeling techniques involve some form of training through which the metamodel is adjusted to best match the training data. There are not many generalizations that can be made contrasting the training of global and local metamodels. Advances in metamodeling have typically required increases in the complexity of the metamodel training procedure with an accompanying increase in the sophistication required of the user. While some complication is justified by improved metamodeling capability, this potential for improved metamodels must not come with an increased risk of obtaining poor metamodels.

5.1.8 Metamodel Selection

The traits that differentiate global and local metamodels outlined in the previous sections imply some guidelines for choosing which class of metamodels should be used on a particular project. These guidelines have been summarized in the following lists. Other authors have given similar metamodel selection guidelines [29, 30]. The first list gives some guidelines which indicate when a global metamodel should be used. This is not a complete list of criteria nor are these absolute rules. However, these guidelines should give good results.

When best to use a global metamodel

- When the underlying behavior of the function to be modeled is well known and the global metamodel can be adjusted (transformed) to match that understanding.
- When the sample data set is noisy.
- When metamodel portability and ease of implementation are very important.
- When using the global analytical form of the metamodel is important.

The second list gives some guidelines which indicate when a local metamodel should be used. As before, this is not a complete list of criteria nor are these absolute rules. These lists highlight the particular strengths of global and local metamodels. If the problem at hand does not show a strong correlation to one of the lists, either a global or local metamodel will probably be suitable.

When best to use a local metamodel

- When the underlying behavior of the function to be modeled is unknown.
- When the sample data set is deterministic (some local metamodels may also be used with noisy data).
- When a good understanding of local accuracy is important.
- When metamodel improvement through adding points (adaptation) is important.
- When very large ranges of the inputs will be used.

This research requires the integration of metamodeling into the design process in a novel manner. In order to maintain generality, it must be assumed that there is no a priori knowledge of the function behavior and arbitrarily large ranges of inputs must be supported.

The metamodels are used as surrogates to black box deterministic analysis codes. Under the circumstances put forth by the goals of this research, it is clear that a local metamodel should be used.

5.2 Basic Local Metamodels

Basic local metamodels are so familiar in engineering that they often are not recognized as metamodels. The simple act of linearly interpolating between two points in a data table is a local metamodel. Some examples of basic local metamodels include linear and cubic spline interpolation and local least squares approximation.

5.2.1 Linear Interpolation

The simplest possible metamodel is that of linear interpolation (with the possible exception of nearest point estimation). In N-dimensional (N-D) linear interpolation, $N + 1$ non-degenerate points in the neighborhood of the point of interest are used to specify a linear N-D function. A Taylor series expansion is used to develop an estimate of the error in the function. The error is shown to be a factor of the second derivative of the underlying function and the point spacing. In practice, the true second derivative is not known and must be estimated by considering more surrounding points. This estimate introduces a higher order error which may be safely ignored for the purposes of error estimation.

While linear interpolation is exceedingly simple to implement and very fast to execute, it has the significant drawback that the resultant function is continuous but not smooth. The first and all higher derivatives are discontinuous at all sample points and across the neighboring point boundaries.

The most difficult part of implementing a linear interpolation metamodel is efficiently determining which points in the neighborhood should be used to build the model (establishing the neighborhood boundary). In general application, this requires an N-D spatial search in an unstructured field of points. Delaunay triangulation generalized to N-D space may be an appropriate technique for this challenge; otherwise, a k-d tree is the canonical data structure for this sort of search problem [34].

5.2.2 Cubic Spline Interpolation

Cubic splines are a form of interpolation where a globally smooth function is built from locally cubic parts [35]. Like linear interpolation, choosing the basis points to use in the computational stencil for a high-dimensional cubic spline is a complex problem. For a cubic spline, the problem is made acute by the increased requirements for the number of basis points. In practice, this regional basis information is generally not stored with the model, instead it is usually re-established as needed.

5.3 Advanced Local Metamodels

Another more complex class of local metamodels exist. These models have been developed to address some of the pitfalls encountered with more traditional approaches to metamodeling including the need to establish a mesh connecting the sample points. These include radial basis functions, neural networks, Kreiging, and Gaussian processes. It has been shown that all of these advanced local metamodels are special cases of a Gaussian process [36], and as the most general case, the Gaussian process will be the primary focus of investigation in this research.

Gaussian processes are a statistical technique which may be used for the regression of data. They may be viewed as the infinite continuous extension of various parameterized models. As such, they do not rely on adjusting parameter values or even probabilistic distributions of parameter values; instead, they rely on the training data and hyperparameters which are tuned to embody assumptions about the behavior of the data. As output, a Gaussian process metamodel evaluates a probabilistic distribution of functions. Gaussian processes are very good at regressing data with complex behavior while providing a statistical evaluation of the quality of the regression [37]. Gaussian processes are discussed in depth in Chapter 9.

5.4 Gaussian Processes

After first choosing to use a local metamodeling technique (p. 54), a Gaussian process metamodel was selected for use in this research. As an advanced local metamodel, Gaussian

processes have enjoyed considerable interest from the metamodeling and machine learning research communities [38, 39, 40]. However, Gaussian processes have not been widely adopted by the systems design and metamodeling communities; the author has not observed any fundamental technical reasons for the systems design and metamodeling community to avoid Gaussian processes. Potential reasons for this slow adoption are manifold, and are typical of the adoption of any new technique. These reasons include the theoretical learning curve associated with Gaussian processes, the lack of Gaussian process implementations integrated with systems design tools, and difficulty in training and tuning Gaussian process metamodels as embodied by the hyperparameters. For a Gaussian process metamodel to be successful in a systems design environment, many of these shortfalls must be addressed.

CHAPTER VI

INFORMATION IN A SYSTEM DESIGN PROCESS

The systems design process is characterized by a huge volume of information. Many systems design challenges are directly related to handling large quantities of multivariate information. The information is used to make decisions or to build approximations when generating more information would be expensive. This information is typically called data and is directly handled by the systems design process.

In contrast to this explicit data, there is also much information that is implied by the systems design process, for example, the assumptions and ranges that go into a design study, who performed a study, why they performed the study, etc. This implicit information can be called metadata and it is frequently ignored or neglected in the systems design process.

6.1 Systems Design Information

Systems design information, the data and metadata, completely describes a complex systems design process. The bulk data constitutes a record of every component analysis case, both inputs and outputs. The metadata constitutes a record of the system structure and the design process itself.

A systems design information management capability must work within the black box paradigm standard to the complex systems design community. In this context, complex systems are built from subsystems and components represented by black box analyses defined by their inputs and outputs. The black box abstraction is introduced such that systems design techniques can be used for any problem.

The data, a record of every analysis case, provides a set of mapped points across the design space. Mapping a point consists of measuring or quantifying that point by evaluating and recording the response at that point. This map of the design space may be used to perform trade studies, to choose a direction of improvement, or to choose a region of interest.

The mapped points may also be used by a metamodel to build an approximate map of the design space between the known points.

While the data in a complex systems process can constitute a huge quantity of information, the metadata is both more compact and far more complex in structure. The metadata records everything about the complex system and the process of studying the system. This information is vital to understanding the components, the information passed between components, and the way the components are assembled into a complex system. This information also serves as a journal recording the complex system study itself, the choices and assumptions made with their justification as well as the tools utilized in performing the study.

In addition to this metadata already present in (or at least implied by) a complex systems study, comprehensive information management brings forth an opportunity to record additional useful information. This information can provide validation, assurance, accountability, and security for the complex systems design process.

6.2 Information Management

In most systems design activities at present, data is typically handled in an ad hoc manner. Each engineer involved with a part of a project develops his own scheme for managing the data relevant to his subproject. There is no centralization or standardization of information management. If another person working on a team needs some information, they must first go to the engineer to get it. If the engineer hasn't stored the needed information in a suitable form, it must be converted or recreated. Much data created during the design process is wasted. New design problems frequently start over from scratch, and data from past studies is not typically reused.

At present, metadata is even more neglected. Decisions affecting metadata are made throughout the design process. Yet metadata only appears at the end of a study, at which point the final results are documented and history is recalled from memory—not records. In many studies, even this step is omitted.

In contrast to the state of the art, design information should be centrally stored in

a standardized form. This information should be made available to global collaborators. Global access to information brings risk to the data; the information may become a target of a malicious hacker, or an authorized user may make a mistake corrupting large amounts of information. The information should be protected through access control and tamper proofing from both accidental corruption and malice. The metadata should be automatically generated and stored alongside the data throughout the design process.

The benefits of a comprehensive approach to information management are manifold. Reduced waste, better accuracy, better collaboration, and more process transparency are just some of the benefits which lead to better designs, in less time, for less cost.

Comprehensive data storage and access will work to eliminate duplicate and wasted cases. Presently, data is not carried forward from one study to another; frequently it is not carried forward from one phase of a study to the next. Cases are often re-run. The data from some cases is never fully utilized. Data use is often limited to an individual or at most a small group, and collaboration is greatly restricted. A centralized data store can solve these deficiencies.

The benefits from comprehensive information management of metadata are perhaps even more pronounced. Making the metadata transparent and available throughout the process will aid in understanding and acknowledging assumptions and limitations, which will facilitate improved decision making. Analysis components will be reused and improved over time rather than re-created for every study. Comprehensive metadata management can lead to data assurance, tracking, and traceability; when problems arise in the data or design process, they are easier to track down and fix. Metadata can be recorded to monitor the subtle changes in tools, and to provide tool validation, test, and training.

In order to achieve these goals, an approach to information management must meet certain criteria. The data must be universally available. The data must be organized in a manner compatible with the black box paradigm of complex systems design. The structure of the database must be general enough to handle any problem. The data and metadata must be stored in parallel. Metadata creation should be invisible and as automated as possible.

The database should store information about its own state to prevent tampering or accident. Analyses and metamodels should be able to be tested and validated to benchmarks stored in the database. Information security should be built into the system, with access controls protecting the information and checks to detect change. Implicitly trusting users has proven to provide inadequate security. Explicit security must protect from both malice and accident; a mistake by a novice user can cause as much damage as an attack on the data.

The system should provide accountability to trace decisions to a group or an individual. This provides another component of the assumptions and reasoning built into the information. It also provides a link to identify exactly who and what you are trusting when you use the data.

Access controls protect the data from unwanted users, but also allow varying permissions to access subsets of the data. For example, a metamodel creator can lock down the state of a metamodel such that other users can not alter it. However, the other users can still have access to the metamodel for query, and can use it in their studies.

6.3 Data Architecture

An overview of a data architecture suitable for complex systems design has been included as Figure 3. Understanding this architecture requires understanding some terms used by the database community. In this diagram, groups of data are called entities. Entities are depicted by a box labeled with a descriptive name.

Relationships between entities are depicted with lines connecting the entities. Relations can be one-to-one or one-to-many. A one-to-one relation connects two specific entities while a one-to-many relation connects a group of entities to a specific entity. A "to-many" connection is represented with an arrowhead on the end of the line. In this diagram, many-to-many relations are simulated by creating an intermediate "pair" entity and appropriate one-to-many relations; the pair serves as the "one" joining the two "many"s in many-to-one-to-many relationship. When a pair table is nontrivial, it is depicted as an entity box with rounded corners; trivial pairings are implied by a line with two arrowheads. Situations

where a relation can connect to a variety of entities are depicted with broken lines to each of the compatible entities.

Any variations from these conventions will be described as they are encountered. Figure 3 is decomposed and explained in the following sections; in each such section, a highlighted version of Figure 3 is presented to identify the entities and relations under discussion.

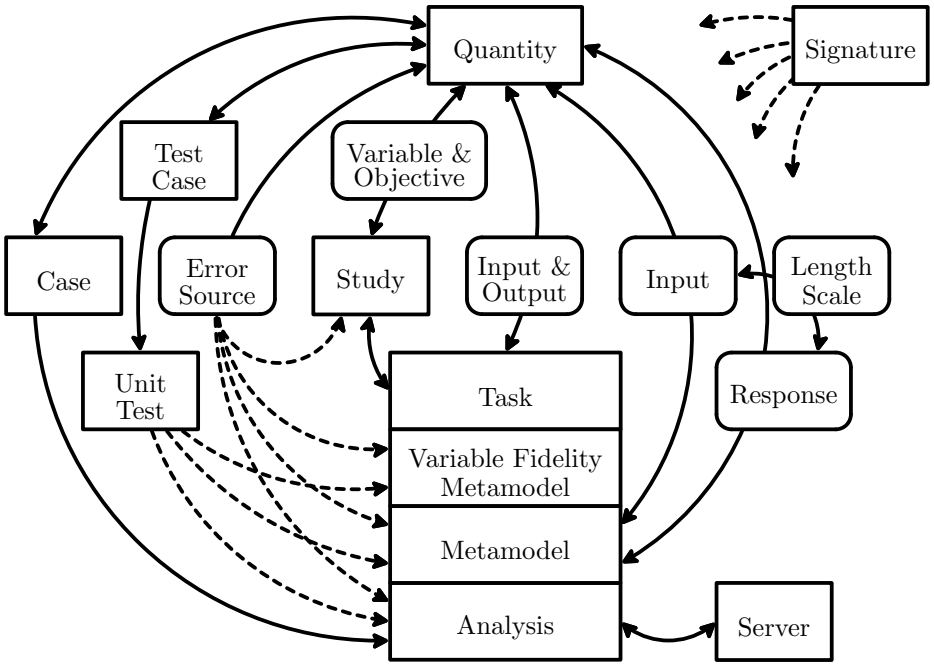

Figure 3: Data architecture overview

Complex systems design treats subsystems and components as functional black boxes. The task entity is used to represent a functional black box, as shown in Figure 4. A black box takes a series of inputs and produces a series of outputs. These inputs and outputs are in turn represented by quantity entities. The tasks are related to quantities through input and output pairings.

A task can be thought to define an interface without specifying any information about the implementation contained within the black box. An interface is often thought of as a contract. Any implementation that adheres to the contract (implements the task) can be used in place of any other.

Figure 5 highlights the various entities which implement a task interface. For simplicity,

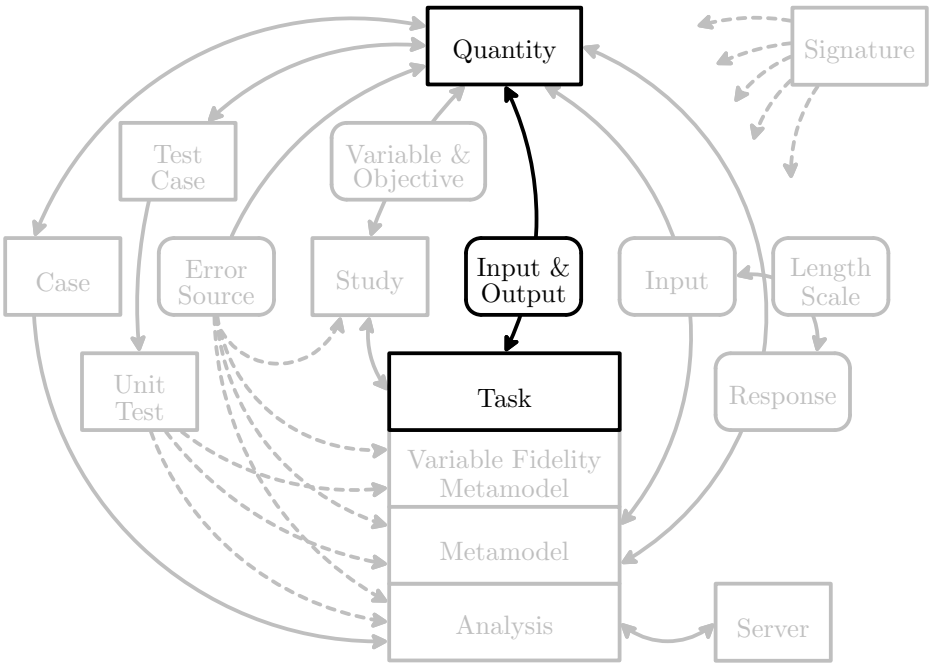

Figure 4: Task entity architecture highlighted

the relations between these entities are not shown; instead they are placed in direct contact to represent their close hierarchial relationship. The entities that implement a task interface are a variable fidelity metamodel, a metamodel, and an analysis.

From the bottom up the entities which implement a task are as follows: an analysis which represents an actual computer program that implements the task interface, a metamodel which represents an approximate surrogate of an analysis, and a variable fidelity metamodel which is a collection of other metamodels that can be combined into a hybrid approximation using variable fidelity techniques.

Figure 6 highlights other entities required to support an analysis. Analyses are hosted and executed on a server. Each execution of an analysis results in a case. Each case is associated with all of its input and resultant output quantities.

Figure 7 highlights the other entities required to support a Gaussian process metamodel. If another metamodel type were to be implemented in the data architecture, a similar set of entities and relations would need to be devised to accompany it. The Gaussian process metamodel requires information pertaining to the inputs and the responses at large as well as

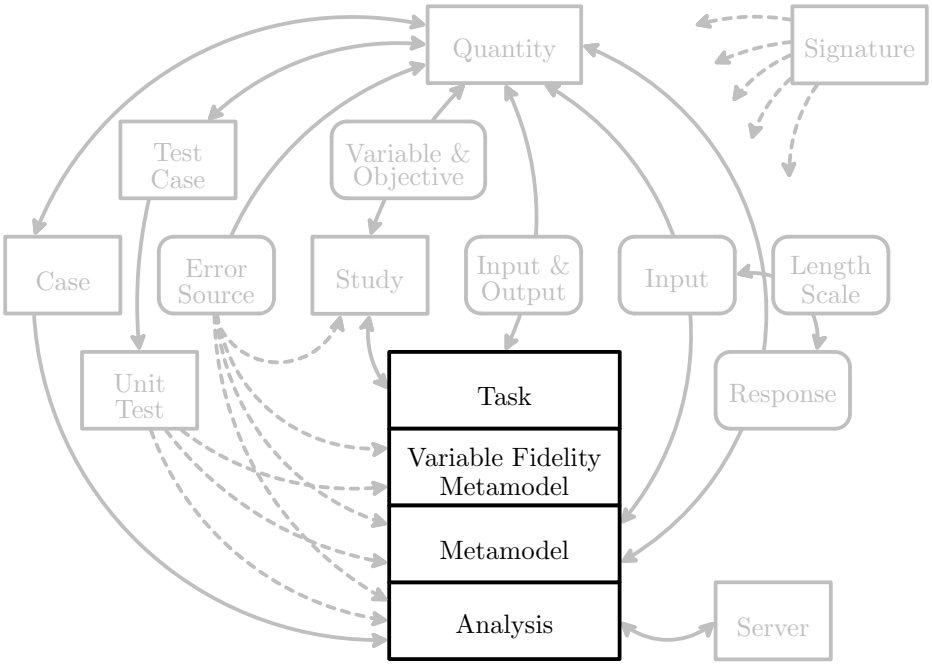

Figure 5: Entities that implement a task interface highlighted

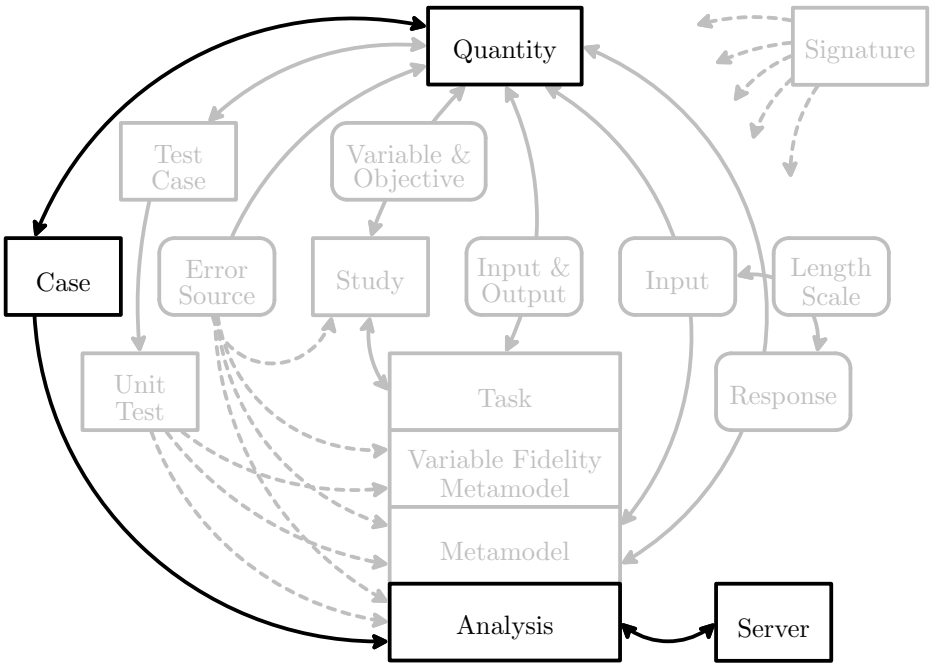

Figure 6: Analysis entity architecture highlighted

information relating each input to each response. This information includes identification of the training set, any scale factors used in normalization, and the hyperparameters pertaining to the trained Gaussian process.

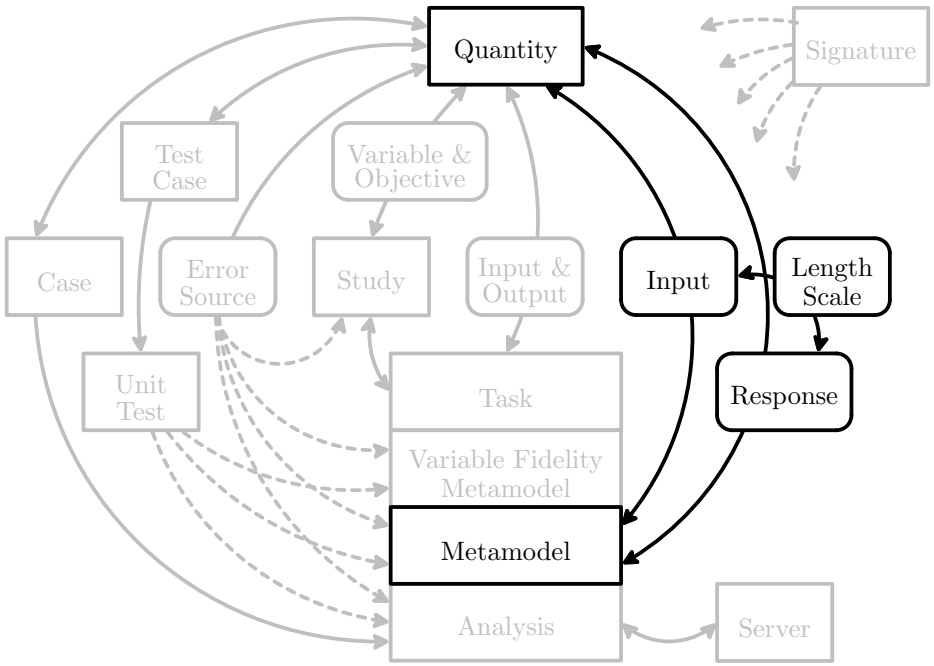

Figure 7: Metamodel entity architecture highlighted

Information security and assurance is first addressed in the data architecture by the unit test entity highlighted in Figure 8. A unit test can be created for any entity which implements a task. A unit test is accompanied by a series of test cases which are associated with the values of the inputs and outputs resultant from the test cases.

Unit testing is a relatively new concept in software development practice [41]. Unit tests are typically simple programs integrated directly with the code which they are intended to test. The tests are developed with the code and are meant to fully exercise the code. By thoroughly testing the building blocks, unit testing attempts to eliminate the explosive combinatorial problem of exhaustively testing large complex applications. Whenever the code is changed, the unit tests are run to ensure proper functionality. When a bug is discovered, a unit test is developed to trigger the bug. The new test protects the code from regression.

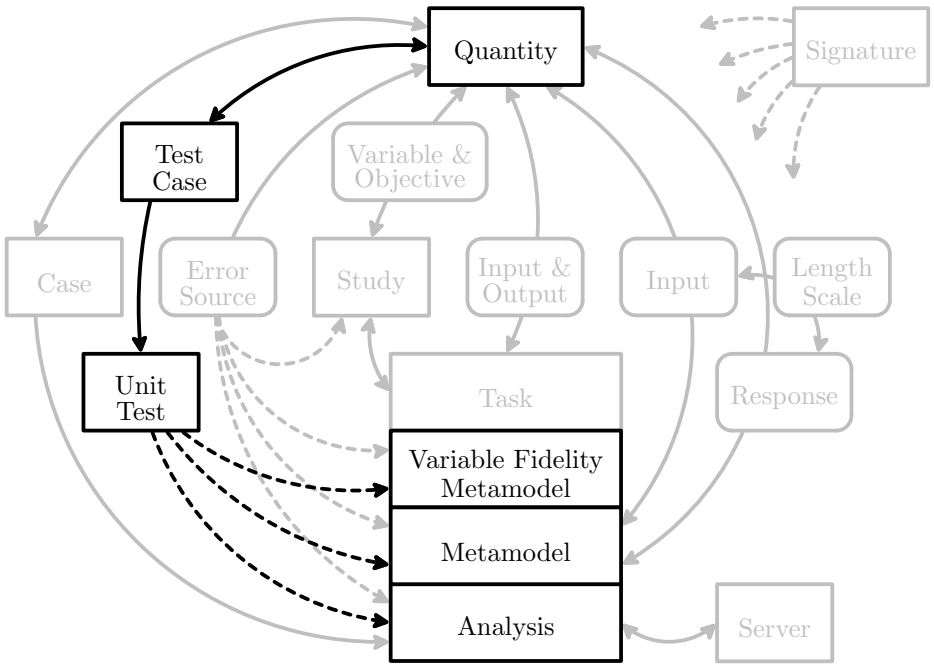

Figure 8: Unit test entity architecture highlighted

In this context, a unit test is a slightly different concept inspired by the unit tests of software development. Here, a unit test is a recorded action. It is attached to an entity which implements a task; in this data architecture, task implementors are the only verbs, all the other entities can be thought of as nouns. The record stores all of the information needed to recreate the action and all the information needed to verify that the action's results have not changed since the unit test was created. Any unit test can be re-run at a later time, the results can be compared with the stored reference results, and any discrepancy can be identified.

Unit tests can be used for validation, diagnostics, tutorials, and documentation. As validation, a unit test will show if the results of an action have changed. Changes can arise from code changes or from data alteration in the repository. As diagnostics, a unit test can exercise an action throughout its range of operation and ensure the record matches intuition. As a tutorial, a unit test can provide pedagogical examples. These examples can include nominal behavior as well as worst-case situations providing insight into the expected behavior as well as how that behavior is achieved. Finally, as documentation, the unit test

records what is expected of an action, how to use it, how it performs, where it is known to fail, etc.

The assemblage of tasks into a system and an investigation of the system are represented by a study entity highlighted in Figure 9. The system is built of various subsystems or components, each represented by a task. The study has various system level input (variables) and output (objectives) quantities.

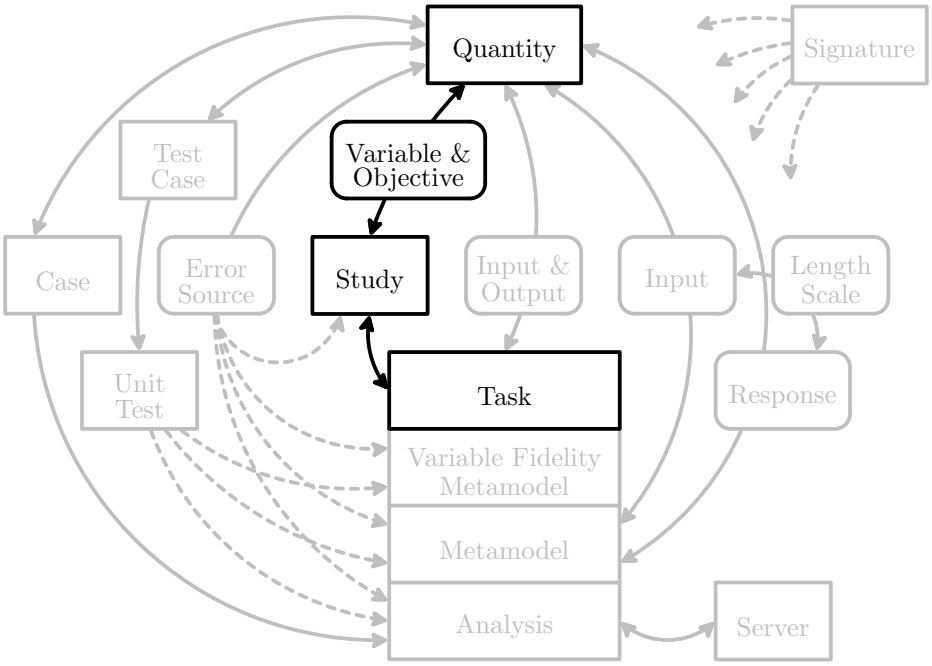

Figure 9: Study entity architecture highlighted

In this data architecture, error can only be introduced to a quantity by the system assemblage or by the actions involved in implementing a task. Various sources of error can be attached to these entities as highlighted in Figure 10.

Information security and assurance is further addressed in the data architecture by the signature entity highlighted in Figure 11. As indicated by its disconnected broken lines, a signature can be applied to *any* other entity. A signature will contain an encrypted code ensuring that it has not been tampered with since its creation by the signer. The signature will also contain a unique fingerprint code which can be used to verify that the signed object has not been tampered with. This fingerprint will most likely be some sort of checksum

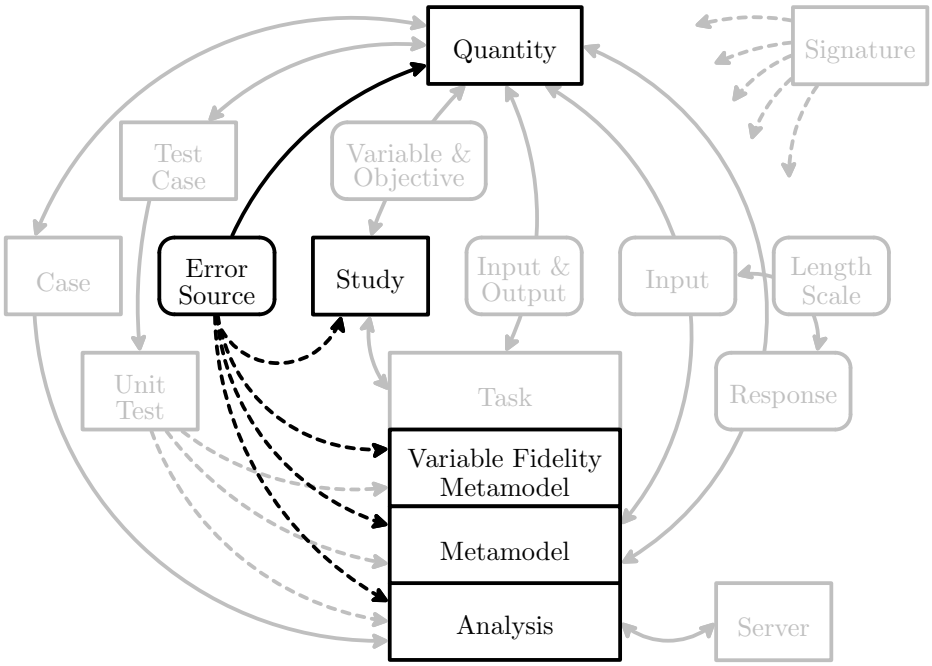

Figure 10: Error entity architecture highlighted

hash function such as SHA-1 [42].

The use of a digital signature allows the data architecture to protect any entity from change, either through accident or malicious tampering. If you trust the signer, then you trust the signed entity. A signature can be applied to an individual entity such as an analysis, or to a group of entities, such as a series of cases executed by an analysis. Of course, signatures can also be applied to the unit tests discussed above, strengthening their value as a security mechanism.

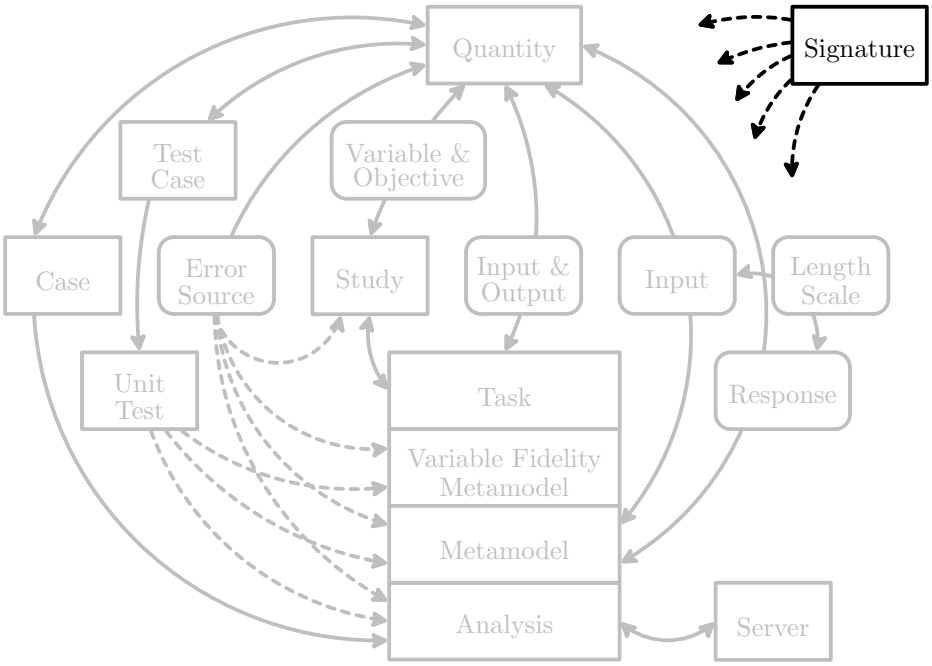

Figure 11: Signature entity architecture highlighted

CHAPTER VII

ARCHITECTURE

Creating a fidelity trade environment tightly integrated with the metamodeling process suggests and requires a significant change to the design architecture. Simultaneously, it is desirable to integrate the new approach as seamlessly as possible into the existing systems design environment such that existing investments may be leveraged with the new capability. For example, designers may already be familiar with certain design space exploration tools, or may have customized them, and the analysis tools may already be integrated into the environment. This represents a significant intellectual investment that one can not afford to discard lightly.

7.1 Typical Design Environment Architecture

Design environments typically abstract the analysis away from the design problem. A standard protocol is established to facilitate communication between the design space explorer and the analysis tools. This protocol defines how quantities are set and read from the analysis. It also defines how the analysis is started, and how the system is notified upon completion. Such a system is illustrated in Figure 12. For the purposes of this discussion, and at this level of abstraction, all design environments may be viewed in this manner [43, 44]. Most design environments adopt a nomenclature where the analysis code is a server, and the design space explorer is a client. The client makes a request to a server which acts on that request and returns the results according to the predefined protocol. This terminology will be used herein. In this approach, the communication between two analyses (servers) is handled by the design space explorer (client).

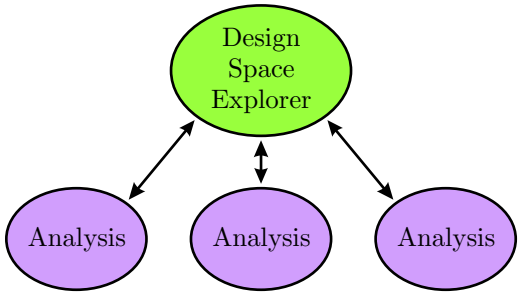

Figure 12: Conventional design architecture

Design Space Explorer Here the client (design space explorer) can be any sort of design space exploration tool including a visualization environment or a domain spanning, stochastic, or path based optimizer. Any of these tools operate by repeatedly investigating the design space at a series of locations. These tools may be written to use a common interface that is familiar to the designer.

Analysis Here the server (analysis) can be any computer code which has been properly wrapped as a black box to function in the client/server environment. The server system then takes care of executing the code to provide results at a desired point. The job of wrapping and maintaining the analysis tool itself may be left to the disciplinary expert; the designer does not have to become an expert in every aspect of analysis, or in every code.

Remote Procedure Call Protocol The arrows in Figure 12 represent the remote procedure call (RPC) protocol used by the design environment. The RPC protocol is the dialect and means of communication used by the client-server environment. It handles the communication to set a server to run at a point of interest, executes the server, waits for completion, and then communicates the results back to the client.

Metamodeling The architecture depicted in Figure 12 has no integrated provisions for metamodeling. Metamodeling must either be performed outside the environment in an ad hoc manner and then wrapped as an analysis, or it must be integrated with the design space explorer, possibly impacting the normal design space exploration activities.

Error Analysis In this architecture, system-wide error analysis should be performed by the integration tool, the design space explorer. The client is the only component that is aware of the system structure. No tools known to the author perform error analysis on a system level.

7.2 New Design Environment Architecture

In order to seamlessly integrate the new techniques into the design environment, a strategy where the entire metamodeling system is introduced as an intermediate layer is adopted. There, it takes on both client and server behavior. The metamodel appears to the design space explorer as an analysis code (server). Simultaneously, the metamodel appears to the analysis codes as a design space explorer (client). This approach is depicted in Figure 13. The existing RPC protocol is used for all communication on both sides of the metamodel.

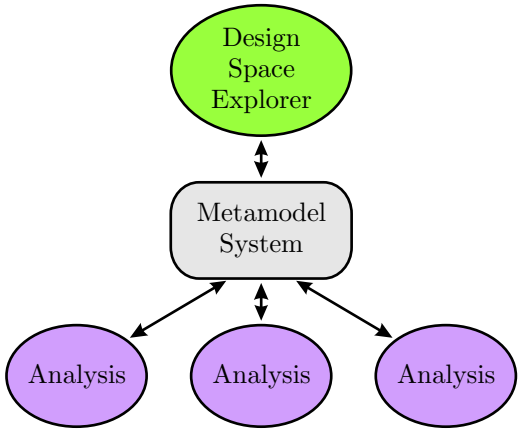

Figure 13: Integration of metamodeling with the design architecture

It is simple enough to say that a system meeting all the outlined requirements may be inserted into a design environment in this way, but when developing such a system, it makes sense to break down the problem into various components according to a division of labor. The function of these components will be similar to the steps in the conventional metamodeling process augmented with systems error analysis and management and will flow rather naturally from the functions of the environment laid out previously.

Figure 14 depicts in more detail the computational architecture integrating the metamodeling system and the error propagation capability with the systems design environment. In this arrangement, the system visualization and error propagation interface lie to the side of the core client-metamodel-server pathway.

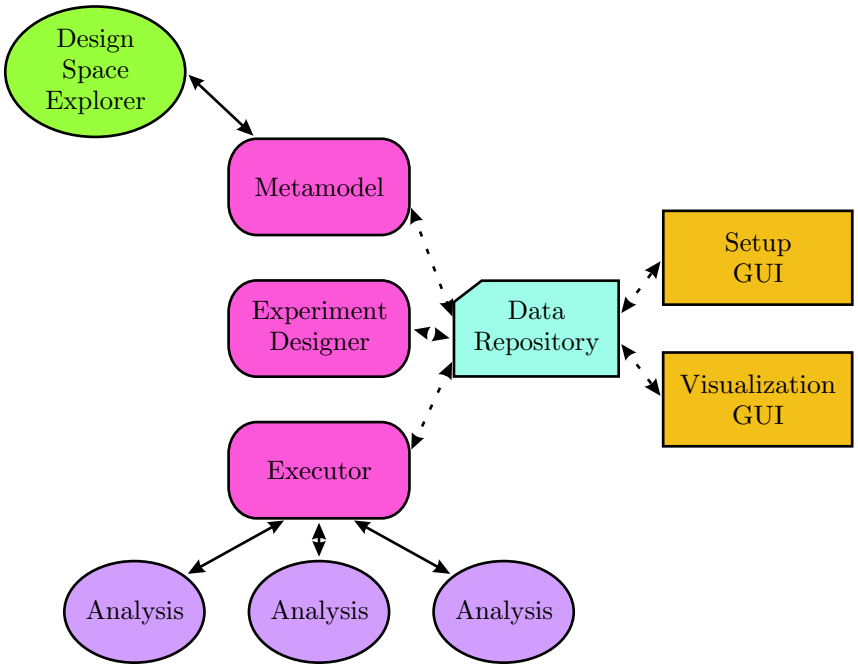

Figure 14: Details of the new architecture

Each of these components has a purpose, and interacts with the other components in a certain way in order to accomplish its job. The components are classified as either primary components (those that primarily provide design functionality) or support components (those that primarily provide infrastructure).

7.3 Primary Components

The core components of the integrated design architecture give the technique its error management, metamodeling, and data management functionality. The visualization GUI provides an intuitive interface to make decisions concerning error and design, the metamodel provides a high efficiency system model, and the data repository manages the complex systems data and metadata.

7.3.1 System Error Visualization Environment

The visualization GUI is the visible arm of the fidelity trade environment, directly satisfying the need for an error management interface. An interactive interface allows quick diagnosis of the design problem and provides intuition into the behavior of the design problem. With such an interface, an understanding of the impact of error, its propagation, and minimization is gained in a way that would not otherwise be possible.

The visualization GUI interacts with the user as its primary source of input; it dynamically responds to the designer's inquiries. It communicates with the data repository through the repository API to obtain information about sample points and design studies.

The visualization GUI has a built-in version of the metamodel to facilitate rapid modeling of system behavior. Various customary design space depictions are provided; these are augmented with depictions of propagated error including metamodeling error as well as any user-estimated error. Also, an interface to interact with error estimates and perform error allocation studies is provided. In addition to a global perspective on error, the GUI also provides an error summary capability for the point of interest highlighting the way error interacts through the problem and a breakdown of the sources of error.

7.3.2 Gaussian Process Metamodel Component

The metamodel acts as the server side of the metamodeling architecture as depicted in Figure 13. It directly fulfils a need for fast component analysis by calculating the best estimate of a response at a point of interest.

As an analysis server, the metamodel receives and responds to requests from the design space explorer through the appropriate RPC protocol. The metamodel queries the database for relevant sample points through the data repository's application programming interface (API). The metamodel also retrieves setup information from the database.

The metamodel uses relevant sample points obtained from the data repository to infer behavior away from the sample points using a local metamodel.

7.3.3 Data Repository

At the core of the architecture depicted in Figure 14 is a central repository for all data, directly fulfilling the information management need laid out previously. In this data store, all inputs and outputs of interest for all cases ever run are retained, ensuring data accumulated from past design studies is not wasted when a new study is started. In addition, metadata describing design studies, various analyses, and available computational resources are stored. Further, metadata to provide data assurance, security, documentation, and traceability is generated and stored. The data repository is the only component of the architecture which can store information for later retrieval; all other components rely on it for their information needs.

The data repository responds to requests to store and retrieve information. Requests are placed (and fulfilled) through its protocol, locally or over a network, by computer programs through an API, or manually through an interface. This means the data is inherently accessible to an entire network, enterprise, or the world. The components of this design architecture will access the repository through its API.

The data repository is a purpose-built off-the-shelf program designed to efficiently store and retrieve information.

7.4 Support Components

The support components of the integrated metamodeling architecture provide vital infrastructure to the functional core components. The experiment designer, executor, and setup GUI store data, arrange cases, execute cases, and interactively set up complex design problems.

7.4.1 Experiment Designer

The experiment designer helps the designer decide what points need to be run and queues them for execution. These points provide the bulk of the training data for the metamodels.

The experiment designer uses the repository API to submit queries to the data repository to obtain sample points in the design space. It provides a variety of user interfaces suitable

for queueing points one at a time or in batch. The program can be accessed directly or from an external program. The experiment designer queues cases by placing them into a run queue stored in the repository.

This communication is done through the database protocol. Functionally, the experiment designer balances these many influences and builds an execution queue. When the queue is prepared, it is stored in the data repository.

7.4.2 Executor

The new design architecture described in Figure 13 requires a metamodel system to have both client and server behavior. The client behavior initiates and monitors the execution of programs by the analysis server. The executor component monitors computing resources and a work queue, making sure all cases get run in an efficient manner. This task is inherently parallel, so the executor can be used to increase the parallelism of a design environment that was originally serial.

The executor obtains the run queue from the database through the database protocol. It communicates with the analysis servers through the RPC protocol to initiate and monitor cases. When a case is completed, it inserts the results into the database.

The executor constantly monitors all available computing resources, all executing codes, and the run queue to ensure highly efficient operation. As a resource comes online or is made available by the completion of a case, a new case is spawned to utilize the resource. This task is inherently dynamic and is particularly well suited for a threaded implementation. The executor also employs a level of fault tolerance to handle the inevitable: cases may crash and servers may go offline.

7.4.3 Setup GUI

The graphical interface of the setup GUI allows interactive scans to be performed, giving the user a choice of available computing and analysis resources. In this way, complex systems metadata is interactively created with minimum effort, time, and error.

The setup GUI scans the network for analysis codes and interrogates the analysis codes to obtain their inputs and outputs via the framework's RPC protocol. Information not

obtained automatically is entered into forms by the user. The Setup GUI stores all configuration information in the data repository through the repository's API. Configuration information can also be retrieved from the repository.

7.5 Software Implementation

As a new systems design technique, the implementation of the architecture depicted in Figure 14 is largely arbitrary. Of course, how the tool is created has many ramifications and therefore requires careful planning, yet other circumstances could always lead to a different implementation. The details discussed in this chapter are central to the successful development of a tool but not to the success of the technique.

When deciding how to implement the technique, it is important to consider the important implementation traits implied throughout the earlier chapters. These include seamless integration with existing systems design tools, remote access to data, remote access to diverse computing resources, use of off-the-shelf technologies, and modular design. The central tenet to these traits is that the implementation should not restrict the designer any more than his choice of a systems design environment already has.

While the implementation decisions described below are presented as individual choices, they are in fact tightly related and the decisions were made holistically.

7.5.1 Programming Language

When the development of a new computer program commences, one of the first design decisions made is the choice of programming language. Today, there are a myriad of general-purpose (and specialized) compiled and interpreted languages generally available while new languages are constantly being developed. At the lowest level, no matter the language, all programs come down to the same commands executing on the processor. Different languages exist to explore different approaches to expressing the problem in a human-compatible form.

This variation leads to the strengths and weaknesses inherent to each language in solving a particular kind of problem. The Fortran programming language [45] and Matlab application [46] are well suited to programs heavy in mathematics. The C programming language [47] was intended for systems level development while C++ [48] was developed

with applications in mind. Some scripting languages are very good at working with text or network applications. When starting a project, the choice of programming language should be made by identifying the best tool for the job, not simply by identifying the language most familiar to the developer.

The Java programming language [49, 50] was the clear choice for this project for a variety of reasons: Java is freely available and designed to be portable across platforms, Java is an object-oriented language with built-in support for threads and GUIs, and most importantly, Java provided seamless integration with the design environment and database.

7.5.2 Framework Foundation and RPC Protocol

As discussed earlier, the new integrated metamodeling and error propagation techniques are to seamlessly become a part of an existing systems design environment. Any design environment which may be viewed to operate according to the paradigm depicted in Figure 12 may be used as the foundation for this work. At a later date, it should be possible to choose another environment or even inter-operate with multiple incompatible design environments.

Due to the design of the improved architecture, to some extent the design environment is a commodity. Nevertheless, there are some differentiating factors of potential importance. The environment should be capable of performing a diverse variety of design exploration studies and of integrating with any analysis tool. The environment should be well established commercially and accepted by industry. The environment must provide a well documented and capable RPC protocol and preferably provide an easy to use API.

For the client/server RPC protocol, Phoenix Integration's Analysis Server [43] was chosen. Analysis Server is a leader in the field and meets or exceeds all of the criteria outlined above. Importantly, the Analysis Server protocol is fully documented by Phoenix Integration. Using the protocol, you can use any network-aware programming language to communicate. In addition, Phoenix provides a Java class library to provides an API for communicating through the protocol. Using this library, the details of the protocol are hidden from the developer, thus speeding development. The Analysis Server Java API was

used in the development of the advanced design architecture. Phoenix Integration also supports the discovery of servers on a network through the Internet Multicast protocol [51]. This broadcast protocol was used to allow dynamic interaction between the user and the computing resources available on the network.

7.5.3 Data Repository and Data Protocol

The primary purpose of the data repository is to store information and to respond to requests to add, modify, delete, or retrieve that information. This may be accomplished in a variety of ways. For example, a custom application could be developed using conventional data structures which would be manipulated in memory and stored in data files. This application could stand alone, with a custom communications protocol, or parts of it could be built into every component in the design architecture, and the data files could be shared between the components. Any custom-developed approach comes with significant difficulties including development and testing cost, lack of flexibility, and a close tie between the data and the application. Instead, an off-the-shelf stand-alone product designed to efficiently store and access large amounts of arbitrarily organized information may be used: a database.

Since their introduction in the 1960's, databases have dramatically evolved. This evolution continues today, providing an array of database technologies for an application designer to choose from. Databases may be classified by their data model. Early databases employed the network or hierarchial data models. Relational databases were first proposed in 1970, long before computers could efficiently implement the model. By the mid 1980's, mature relational databases were on the market and have dominated to this day. Object-oriented and object-relational databases are gaining acceptance. Other database technologies continue to advance; parallel, real-time, main-memory, active, temporal, fuzzy, and distributed are just some of the buzzword adjectives commonly heard before *database*. For an in-depth discussion of database technology past, present, and future, see Atre [52] and Piattini and Diaz [53].

Object-oriented databases are capable of storing arbitrary data types, appropriate for computer aided design and manufacturing (CAD & CAM), multimedia, etc. Arbitrary

data types are potentially a powerful feature when working with complex systems design. In contrast, relational databases are only capable of storing primitive data types. Complex entities must be constructed from these primitives, resulting in a layer of translators that must be developed to interface with complex applications. However, the standardization, commoditization, maturity, and availability of relational databases are imploring factors. Furthermore, the limitations imposed by the relational data model provide no impediment as they are consistent with the limitations of the black box analysis paradigm required by client/server analysis frameworks and metamodeling in general.

There are many capable relational databases available today. Standardization of the Software Query Language (SQL) [54] and the Java Database Connectivity API (JDBC) [55] as interfaces to relational databases make the choice between database applications nearly arbitrary for a small-scale demonstration project such as this one. A freely available, stand-alone relational database, MySQL [56], was selected. It has adequate performance, is well supported, is easy to set up and administer by an individual, and works on most platforms. It is an excellent choice for a small-scale research project. In a production environment, migration to a large-scale commercial database such as Oracle, Postgres, or MS Access would not be difficult.

There are a variety of ways to access a MySQL database. For example, there is a command line interface for the direct entry of SQL commands, and a C API to directly access all functionality. The JDBC API built into Java for database connectivity is also supported. JDBC requires a driver to work with a particular database product, and there are multiple drivers available for MySQL. MySQL Connector/J [57] is the official JDBC driver for MySQL, and was used throughout this project.

CHAPTER VIII

SYSTEM ERROR VISUALIZATION ENVIRONMENT

The systems visualization environment is the centerpiece of the fidelity trade environment. While it is built upon the data repository, metamodel, and other components, it provides the systems model and error analysis as well as the decision making interface for error and design. In this chapter the underlying theory of the systems visualization environment is presented and the environment is demonstrated on a notional system. The notional system is built from components made of meaningless functions and is only meant as an example.

8.1 System Visualization Theory

In addition to the error propagation and metamodeling theory discussed in previous chapters, there are several non-trivial theoretical and numerical challenges that must be addressed before developing an integrated systems visualization environment. These include representing and analyzing a complex system and generating the required systems visualization.

8.1.1 Component Ordering

Assuming that black box analysis components are available, and that some knowledge of how the system is assembled is available, deciding on the ordering of components is not obvious.

One classic challenge of systems integration is the ordering of tasks. Given an optimization criterion, choosing the best task ordering becomes a computationally difficult traveling salesman problem, although, for most complex systems, the number of components is small enough to render the problem tractable. However, an appropriate objective function for ordering is not obvious. A simple and intuitive choice (and the one used in this research) is to order the components as to minimize the amount and scope of feedback. This approach generally leads to requiring fewer component executions to achieve convergence. If

feedback can be eliminated, this is clearly the optimal ordering. However, in interesting systems feedback can seldom be completely eliminated. Even minimal feedback is not a clear best choice because the components may have differing run cost or differing amounts of nonlinearity. Both of these situations can cause a net benefit by requiring additional runs of one component at the benefit of reduced runs of another component. This can lead to reduced computational cost or to improved system convergence. More sophisticated tools to address this complex problem exist, including DeMAID developed at NASA Langley [58].

For simplicity, an algorithm to reduce the feedback was implemented [59]. The algorithm chosen is a simple path-searching approach. If feedback can be eliminated, the algorithm will do so. The algorithm is not guaranteed to find an optimal solution. A simple system is shown before task re-ordering in Figure 15. The same system is shown after re-ordering in the first chapter as Figure 1, repeated here as Figure 16.

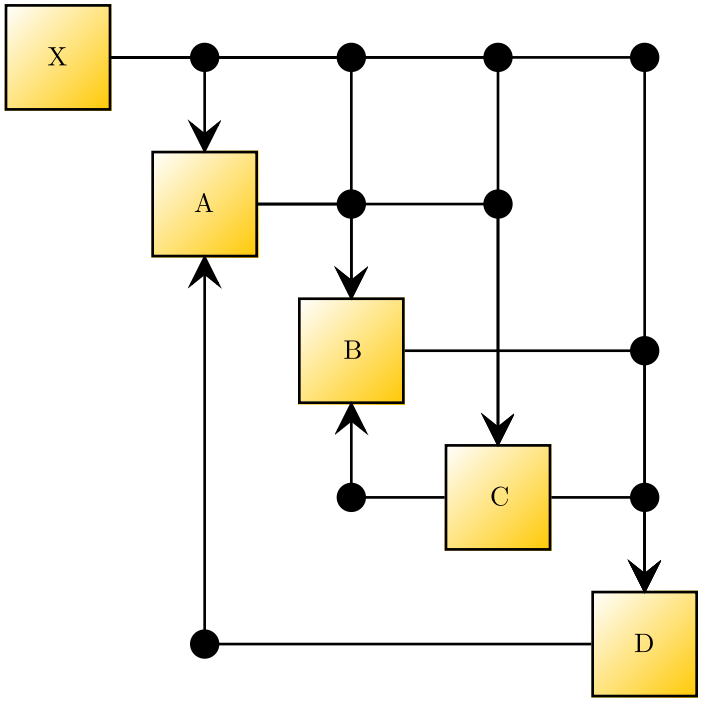

Figure 15: Un-ordered system DSM

The component re-ordering algorithm works by first analyzing each component in the system. If the component depends on no other components, it is moved to the beginning of the system. If the component is depended on by no other components, it is moved to the

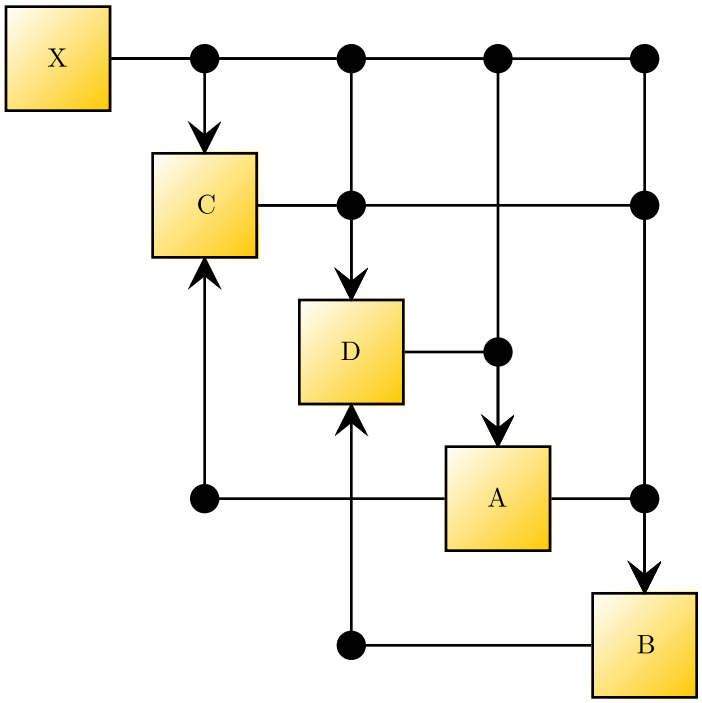

Figure 16: Ordered system DSM

end of the system. After each change the moved component is taken out of consideration for future changes and the system is re-analyzed for components to move.

If the algorithm can not move any components, there must be a coupled loop. The algorithm then follows the data flow in the remaining components until a coupling loop is detected. Ideally, one should strive to find the longest possible coupled loop, however, that subproblem is analogous to a traveling salesman problem. The components which participate in the loop are then nested into a group component (and the re-ordering algorithm is applied recursively to the members of the group). The algorithm then proceeds to shuffle group and other components as possible, stopping when there are no components left to move.

In this research, two passes through the component re-ordering algorithm were made, where the grouped components were "flattened" into an equivalent ungrouped structure between passes.

8.1.2 System Solution

Reaching a compatible state for a set of components comprising a complex system can be cast as finding the solution to a coupled system of nonlinear equations.

In describing the system solution procedure, the system description nomenclature presented in Section 4.6.1.1 will be used and expanded upon. For a given ordering of components, a subset of the component outputs will feed back as inputs to the prior components. For the system, the feedback quantities are denoted $\mathbf{f}^*$. Each subsystem's feedback quantities are denoted $\mathbf{g}_i^*$.

In the context of a systems analysis at a given design point, the design point, $\mathbf{x}$, is a vector of parameters. Parameters do not change as part of the solution procedure. The solution of the system is based on achieving compatibility of feedback quantities. Different solution techniques are possible, so a few will be discussed to give some context.

8.1.2.1 Fixed Point Iteration

In fixed point iteration [60], each component is analyzed in order. Any intermediate quantity calculated by a prior component is used as needed. If a quantity is unknown (because it is specified by a feedback from a component not yet run) a reasonable guess for the quantity is made; the guesses are denoted $\bar{\mathbf{g}}_i^*$. Sometime later in the sweep, these quantities are calculated as outputs from a component; the calculated values are denoted $\mathbf{g}_i^*$. A second sweep through the components is made with the guesses equal to the calculated quantities from the previous iteration; i.e. $(\bar{\mathbf{g}}_i^*)_{t+1} = (\mathbf{g}_i^*)_t$. Convergence is achieved when the residuals are sufficiently small; $|(\bar{\mathbf{g}}_i^*)_t - (\mathbf{g}_i^*)_t| < \epsilon$, with ϵ set to some appropriate tolerance.

Two subtle variations on fixed point iteration are possible. In the first variation, described above, if a feedback quantity is used by a later analysis, the calculated value is used. In the second variation, the guessed value is used even though the calculated value is available.

Fixed point iteration fails to converge for many problems. However, it is extremely simple to implement. Some degree of user control can be placed on the convergence behavior by introducing relaxation. Relaxation is a technique by which damping is introduced to

the update procedure. With relaxation, the update equation above becomes $(\bar{\mathbf{g}}_i^*)_{t+1} = k\,(\mathbf{g}_i^*)_t + (1-k)\,(\mathbf{g}_i^*)_{t-1}$ where k is the relaxation coefficient, $k = 1$ results in no relaxation while $k = 0$ results in a system which will never update. Overrelaxation ($k > 1$) can be implemented to accelerate convergence.

8.1.2.2 Newton's Method

Newton's method for systems of equations is a more sophisticated technique which is appropriate for a broader class of problems [61]. While the application of fixed point iteration to the problem of system compatibility is straightforward, the application of a general equation solving algorithm is slightly different. Solving a system is typically cast as finding the zeros of a system of equations. In this case, the zeros to be found are the residuals $\mathbf{r} = (\bar{\mathbf{g}}_i^*)_t - (\mathbf{g}_i^*)_t$. The problem is to solve the system for a given set of parameters $\mathbf{x}$. Consequently, the residuals are viewed as functions of the guesses of the compatibility variable $\mathbf{r} = \mathbf{r}\,(\bar{\mathbf{g}}_i^*)$.

Newton's method uses the derivatives $\frac{\partial \mathbf{r}}{\partial \bar{\mathbf{g}}_i^*}$ of the function to help guide the next generation's guess. Newton's method can be shown to be equivalent to fixed point iteration with optimal relaxation (or overrelaxation) determined by the derivatives. Newton's method exhibits second order convergence and works very well if the function is sufficiently well behaved and the initial guess is sufficiently close to the solution.

There are two variations of Newton's method for solving a complex system that correspond to the two subtly related variations of fixed point iteration. These variations manifest themselves as a change in the residual function and its derivatives. In this research, the first variant was used; the calculated value of a feedback quantity is used as soon as it is calculated.

The analytical derivative predictions from the Gaussian process metamodel were used to calculate the derivative matrix needed for Newton's method. Because the flow-down variant of the method was used, the derivative matrix is more complex. The chain rule must be applied to the component analyses to build up the required derivative. A recursive algorithm was developed to implement the chain rule to construct the derivative matrix by

following the flow of information dependence through the system.

8.1.3 Linked Views of the System

Systems visualization environments today typically offer a variety of ways to interact with the system. These visualization environments may include a prediction profiler, a constraint diagram, a three-dimensional surface visualization, and others. Each of these may be considered a view of the design space. Each view is best suited to display a certain kind of information and to make a certain kind of decision. The designer can and should change views during the design process as he investigates different facets of the design.

However, even though each view may be available in one application, the state of the system is typically not shared between each of the views. A design variable change in the prediction profiler is not reflected in the constraint diagram. If the designer chooses to switch views in the midst of a decision making process, he must make sure that he transfers all of his settings (decisions) to the new view. This imposes a significant impediment to switching views as required; this weakness becomes more significant as the system description grows more detailed, say through increased dimensionality or the consideration of error.

Instead, as stated previously, every view should be linked such that they always represent the same state of the system. While this linking increases the cost of updating the views to reflect a change in state (every view must be updated instead of just one), it is easily justified by eliminating the user cost associated with changing views during the design process.

8.1.4 Constraint Contour Generation

The generation of constraint curves in a design space constraint diagram amounts to finding a contour line at a single value. This is also sometimes called a level curve problem. In this research, a very simple algorithm was used, where the contour is built by interpolating across a uniform grid of points [62]. This approach is simple and reliable. However, in order to get smooth contour lines, it is necessary to use a fine grid of points. Depending on the nonlinearity of the curves, this can get expensive. An example constraint contour plot based on a fine grid of points is included as Figure 17. The same plot re-generated based on a coarse grid of points is included as Figure 18.

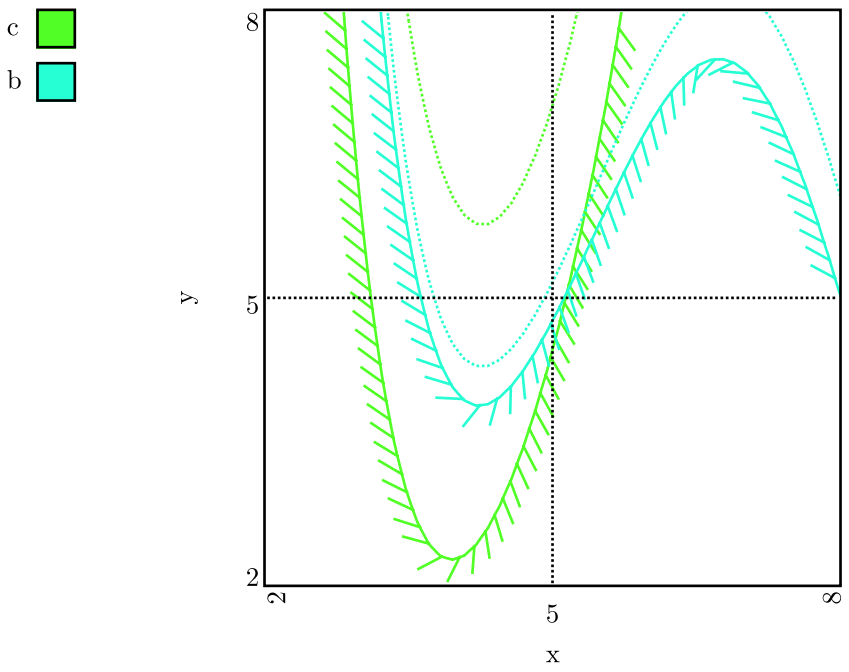

Figure 17: Fine grid constraint diagram

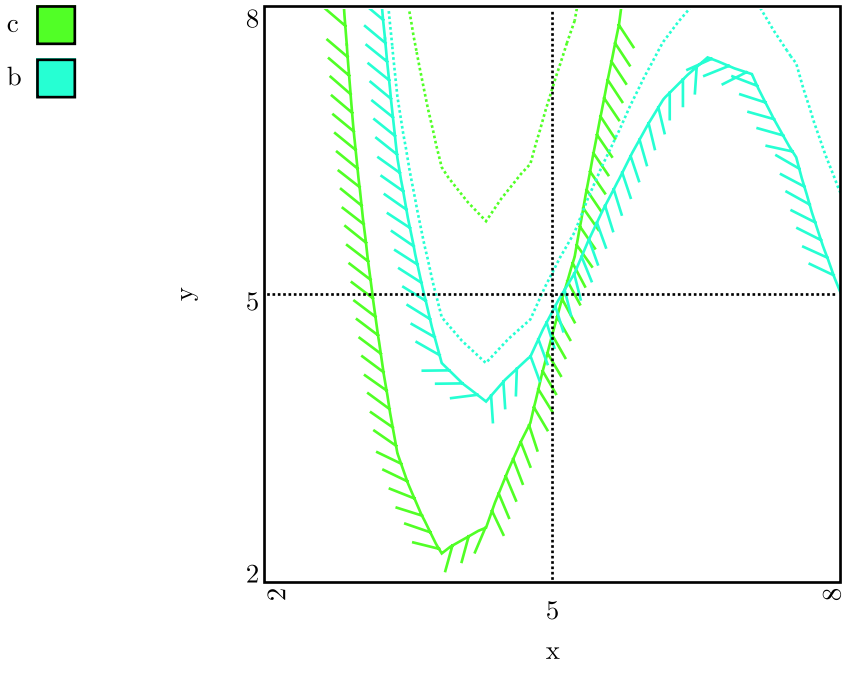

Figure 18: Coarse grid constraint diagram

This contour generating algorithm does not actually generate level curves. Instead, it generates a series of short line segments that fall on the curve. These line segments are produced in no particular order and have no particular orientation. This poses no problem for the traditional contour plot application. However, the constraint problem requires the determination of the orientation of the curve, i.e. which side of the curve is constrained. Rather than perform an orientation test on every line segment produced, the line segments are chained together into a polyline by merging endpoints that fall within a certain radius. An orientation test is performed on each resulting polyline. Working in terms of polylines instead of line segments was also critical for drawing the hatch marks.

The line segment chaining algorithm implemented in this research tool is not perfectly robust. Occasionally, it results in multiple chains when one chain should have been possible. This results in occasional visual artifacts observed as nonuniform hatchmarks along the constraint line. These artifacts occur seldom enough to not be a concern. However, if the chaining step were omitted, the entire curve would be a series of artifacts.

Even with a very high resolution contour plot, changing contour levels is very quick and responsive. However, anything that changes the data constituting the grid of points is expensive. In this environment, that includes a change in the hairlines or the error settings.

8.2 System Error Visualization Environment Interface

The system interface provides various interactive views of the complex system. This section highlights the primary features of the interface. Some additional features and the detailed behavior of the system interface are discussed in Appendix B. A depiction of the systems interface during a typical session is included in Figure 19. The system being investigated has been the example throughout the thesis thus far. It has four components (A, B, C, D) and is a function of three system level variables (x, y, z).

At the bottom right corner of the interface, there is a Design Structure Matrix (DSM) depiction of the complex system. As described in Section 1.4, the DSM represents the system structure by highlighting the information passed between components.

Above the DSM, there is a constraint plot depicting the design space. The constraint

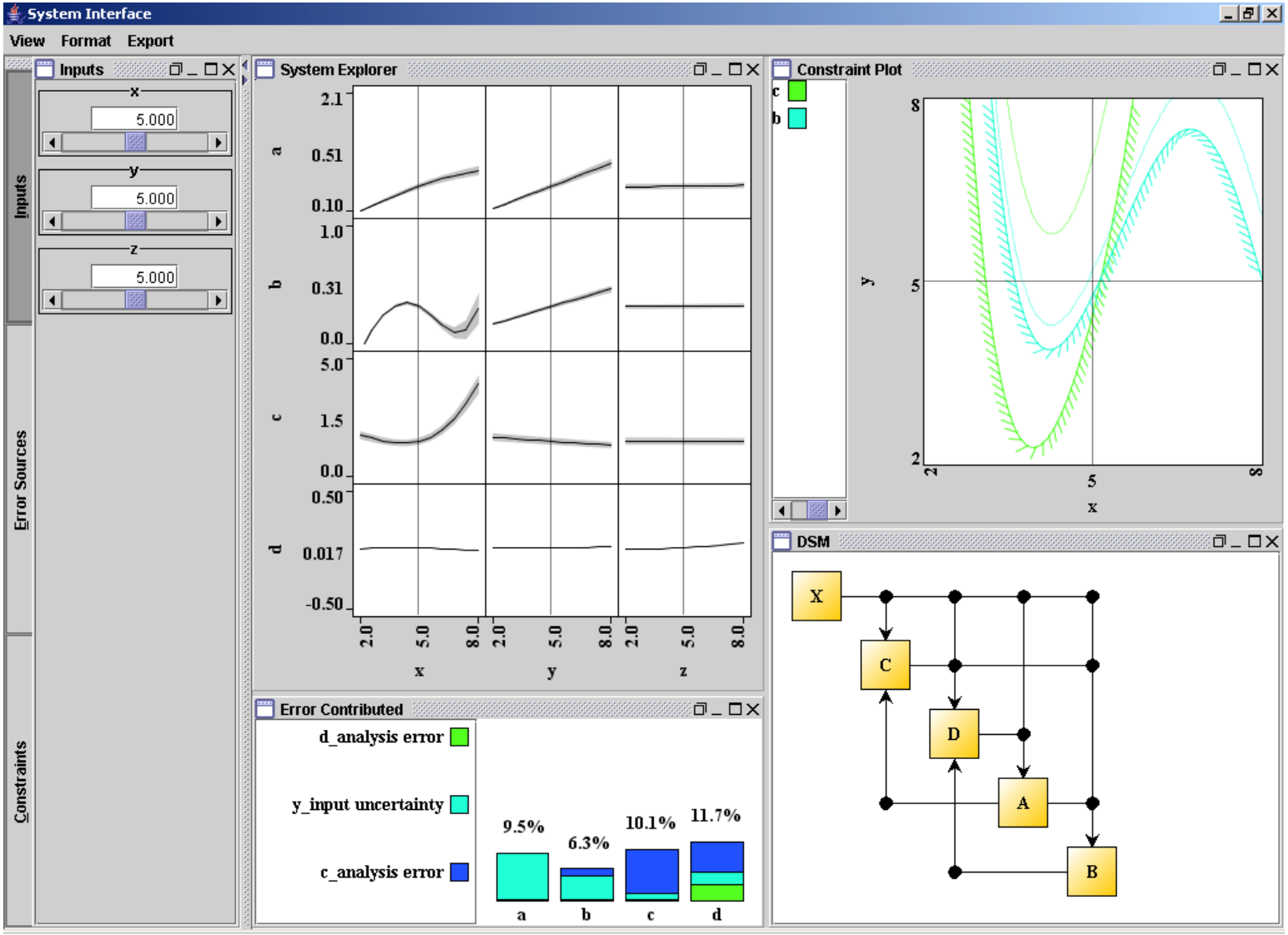

Figure 19: The system interface

plot represents a view of the design space divided into feasible and constrained regions.

To the left of the constraint plot, there is a system explorer depiction of the design space. The system explorer represents the variation of every system response throughout the design space.

Below the system explorer, there is an error contribution display. The error contribution display represents the estimated error at the point of interest for each system response and the contribution to the error by each error source.

To the left of the display, there is a tabbed pane that is currently displaying system input controls. Other tabs which can be selected include error source controls and constraint controls.

The interface has been designed with flexibility as a primary goal. Flexibility allows the interface to be adapted to the decision maker's workflow, rather than forcing his workflow to fit the interface. To this end, the interface can easily be reconfigured at any time. Each of the panels can be resized or rearranged. The panels can be minimized or detached from the main interface; a detached window can be reattached as well. A panel can be added to the tabbed pane and tabbed panes can be detached and reconfigured as needed. The open source JGui library [63] was used to extend Java's Swing library to support these advanced user interface features.

This flexibility is difficult to depict with a few screenshots and sentences, but should be familiar to users of newer computer applications. No effort to demonstrate every possible reconfiguration of the interface is attempted here.

8.2.1 Design Structure Matrix

The DSM provides a straightforward and familiar display of the structure of a complex system. Each of the components is represented by a labeled box. The system level inputs are represented by a special box labeled X. Section 1.4 includes a more through description of a DSM.

Information flows into a box through its top or bottom. Information flows out of a box through one of the sides. Information flow is depicted by a bent arrow with a dot at the

kink in the arrow.

If the user places the mouse cursor over one of the information flow dots and pauses for a few moments, a tooltip appears as shown in Figure 20. The tooltip lists exactly what quantities are communicated through each connection. The tooltip will disappear when the user's attention is diverted to some other part of the interface.

8.2.2 Constraint Diagram

The constraint diagram depicts the feasible portion of the design space by drawing a hatched curve across the design space. The hatched side of the curve represents the region of the space that is infeasible. Each constraint is drawn in a different color with a key displayed to the left of the constraint diagram. A dotted line is drawn in the appropriate color in the feasible portion of the design space indicating the possible change in the location of each constraint when error is considered.

Black horizontal and vertical dotted hairlines indicate the point of interest in the two displayed directions. The hairlines can be dragged throughout the design space to change the point of interest. The numeric value corresponding to the hairline setting is displayed near the axis label. The extents of the design space are also labeled.

In the lower left corner of the constraint plot panel, there is a small unlabeled slide bar. This slide bar adjusts the grid resolution used to generate the contour curves as discussed above. Sliding the bar to the far left results in very coarse contours that update very quickly as shown in Figure 18. Sliding the bar to the far right results in very smooth contours that take much more time to generate as shown in Figure 17.

8.2.3 System Explorer

The system explorer depicts the function space of the system. Each response appears as a row in a grid of plots, and each row is labeled by the response name. Each input appears as a column labeled by the input name. Each input has a specific setting indicated by a vertical dotted hairline at the appropriate place in the range. The setting is quantified by a number at the base of the graph near the column label. Each response value is quantified by a number which appears near the appropriate row label. The ranges of valid inputs

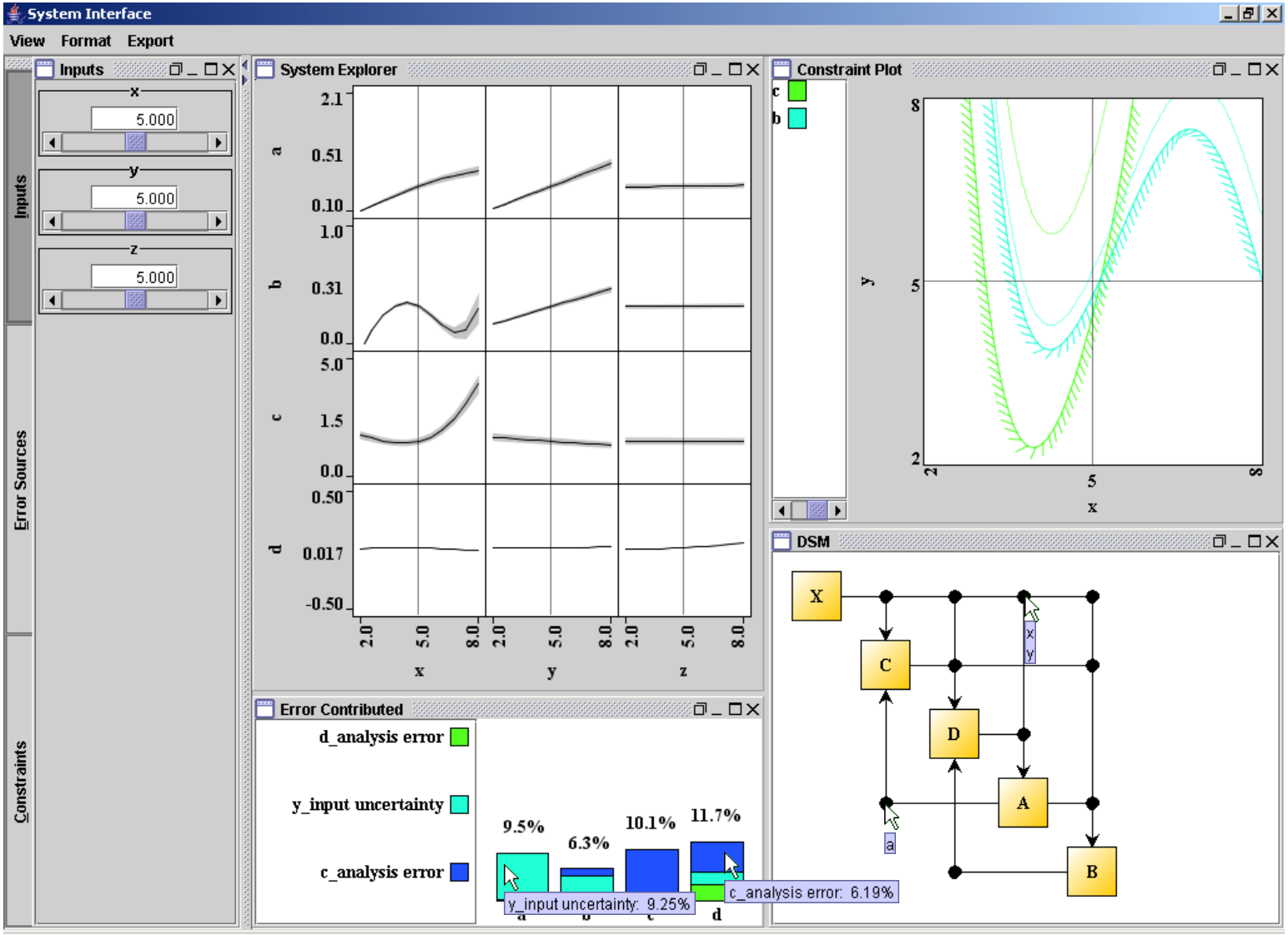

Figure 20: System interface tooltips

and expected outputs are depicted by axis labels representing the minimum and maximum value in each range.

8.2.4 Error Contribution Display

The error contribution display depicts the system propagated error at the point of interest in a stacked bar graph. Each bar is labeled at its base by the quantity it applies to. Each bar is labeled at its top by the magnitude of the total error. The bars are automatically re-scaled to appear in the display.

Each bar is made up of stacked error contributions. Each error contribution is color coded with a key appearing to the left. The height of each contributor is related to the magnitude of its contribution as discussed in Section 4.8. If the user places the mouse cursor over one of the bars and pauses for a few moments, a tooltip appears as shown in Figure 20. The tooltip identifies the error source and lists its contribution to the total error. The tooltip will disappear when the user's attention is diverted to some other part of the interface.

8.3 System Error Visualization Environment Behavior

Some straightforward experiments are presented to build intuition regarding error in a complex system and this user interface. For these examples, the example system is the one shown in Figure 1 which has been used throughout this research so far. The functions representing each of the components are listed below. These functions have no physical meaning and are included for reference purposes.

$$\begin{aligned} a &= \left(2.48374x + \frac{4.746265d^2}{x + 3.2874} + 3.23872dx + [y - 3.48572]\,x \right. \\ &\quad \left. + \frac{[d - 4.34721]\,y}{[x + 5.424]} * 2.3421 + y \right)/40 \end{aligned}$$

$$b = \left(xac + \frac{a}{c^2 + 0.02375}\right)/40;$$

$$c = 2 + \left(x^2\,[x - 5.67834] + ax\,[a - 6.3432]\right)/60$$

$$d = \left(bc\left[c + 0.0345\right] + z^2\left[b^2 + 0.34721\right]\right)/1200$$

8.3.1 Baseline

For the baseline case, the point of interest was set to the center of the design space in all variables, as listed in Table 1.

Table 1: Baseline input

Variable	Value
x	5.0
y	5.0
z	5.0

The response of the system at the point of interest was listed in Table 2.

Table 2: Baseline response

Variable	Value
a	0.509
b	0.312
c	1.469
d	0.0175

Three error sources were introduced and the error levels were set to representative values listed in Table 3.

Table 3: Baseline error sources

Error Source	Value
ςd	0.05
σy	0.075
ςc	0.1

The system propagated error at the point of interest is broken down in Figure 21. Each color coded bar indicates the proportion of the total error due to the corresponding error source. The total error in each response quantity is shown by the total height of the bar.

As a verification experiment, a Monte Carlo approach was used to propagate the error

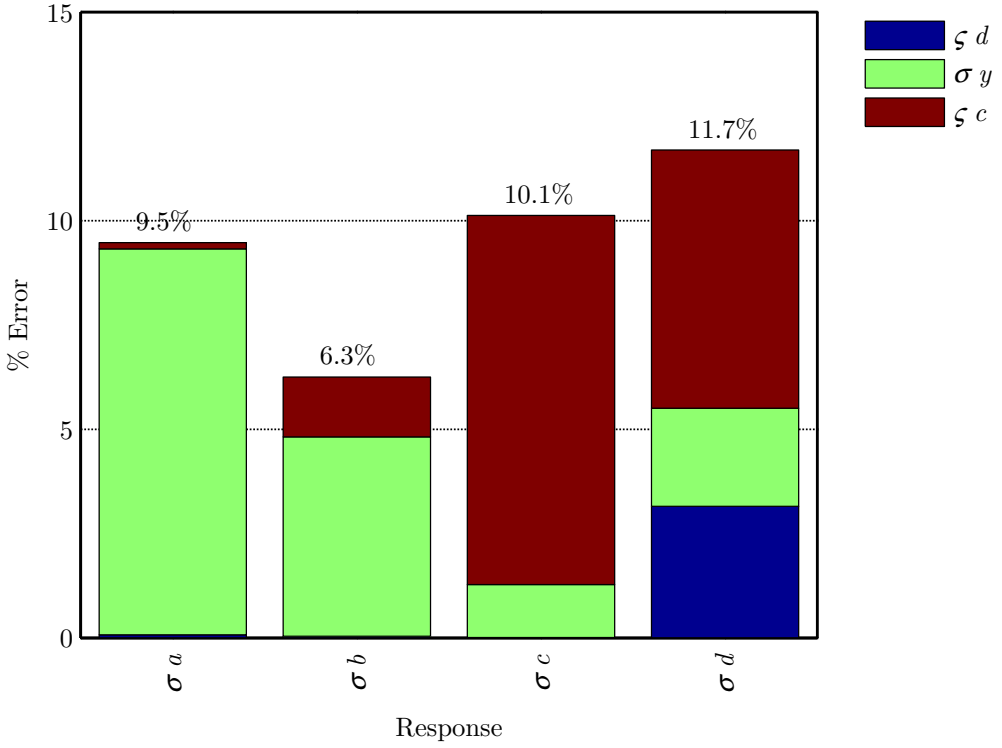

Figure 21: Baseline error breakdown

through the example system. The results of the Monte Carlo analysis were compared with the sensitivity based approach in Figure 22. The blue and red bars represent the results of the sensitivity approach and the Monte Carlo approach respectively. The small error bands at the top of the Monte Carlo bar represent the one sigma error in the result due to the use of a finite sample size.

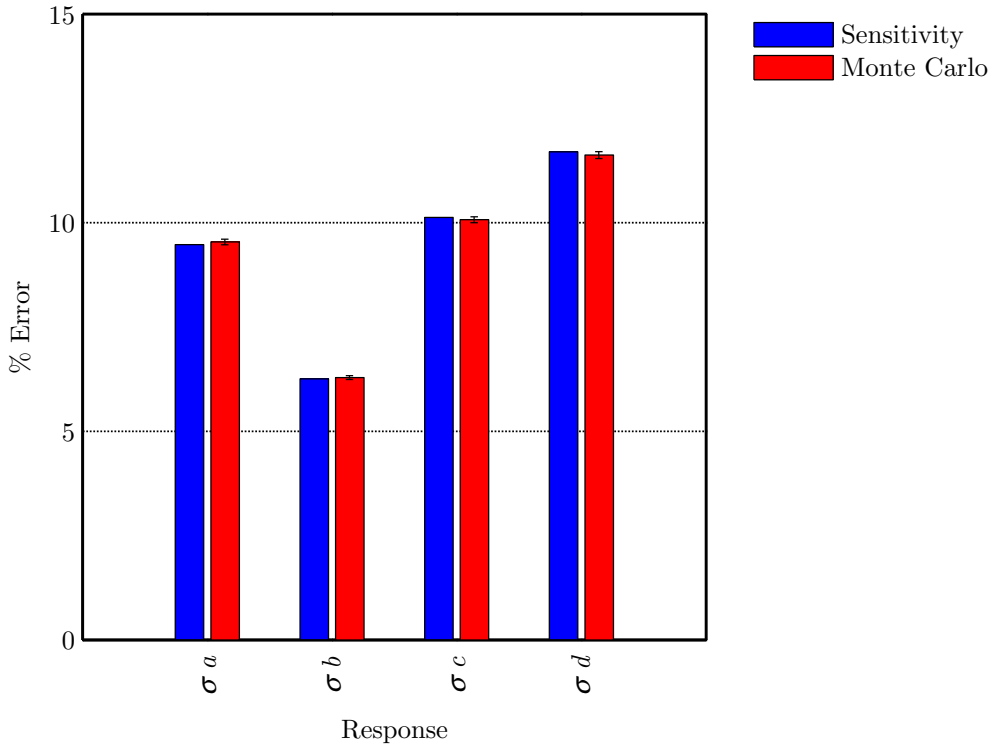

Figure 22: Baseline error verification

For the Monte Carlo study, Gaussian probability distributions with zero mean and the specified variance were assumed for each of the error sources. A random sample of 10,000 cases were analyzed, and the complete system was brought to convergence for every case.

The agreement between the two approaches is excellent, verifying the sensitivity approach as an approximate model of the propagation of error through a complex system. None of the differences in this case can be definitively attributed to the approximations made by the sensitivity approach.

The system response obtained by varying each input variable individually is presented

in Figure 23. The gray bands represent the system propagated error in each response throughout the design space.

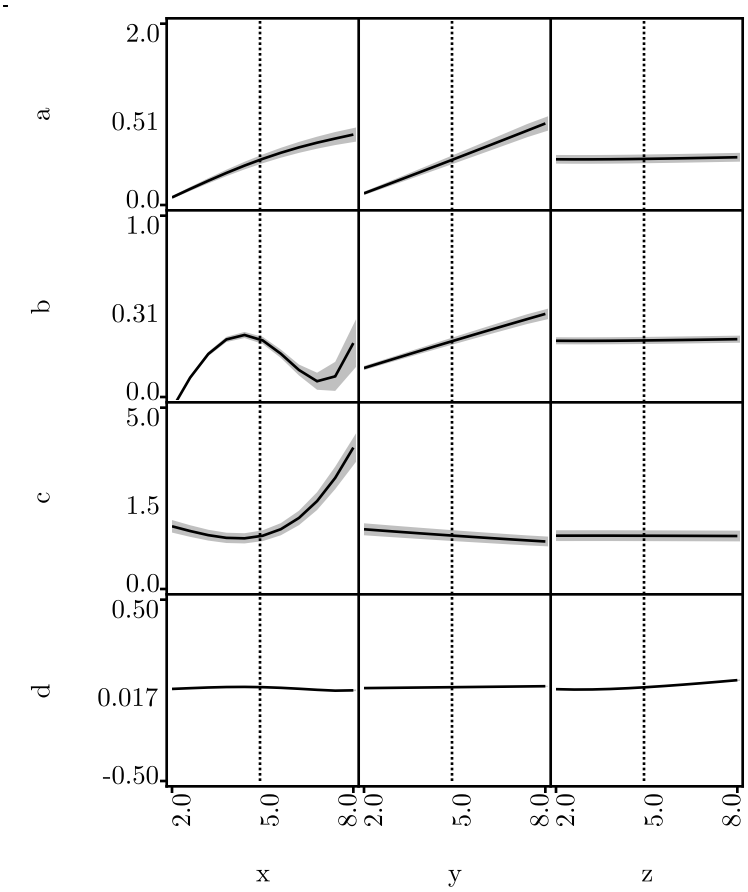

Figure 23: Baseline system response

Two constraints were applied and the constraint values were set to reasonable levels listed in Table 4.

Table 4: Baseline constraints

Constraint
$c < 1.5$
$b > 0.3$

The constraint diagram corresponding to these constraint levels was included earlier as Figure 17, repeated here as Figure 24. Each constraint is identified by the color-coded key. The infeasible side of each constraint is indicated by the hatch marks along the constraint

curve. The position of the curve with consideration of propagated system error is indicated by the dotted line.

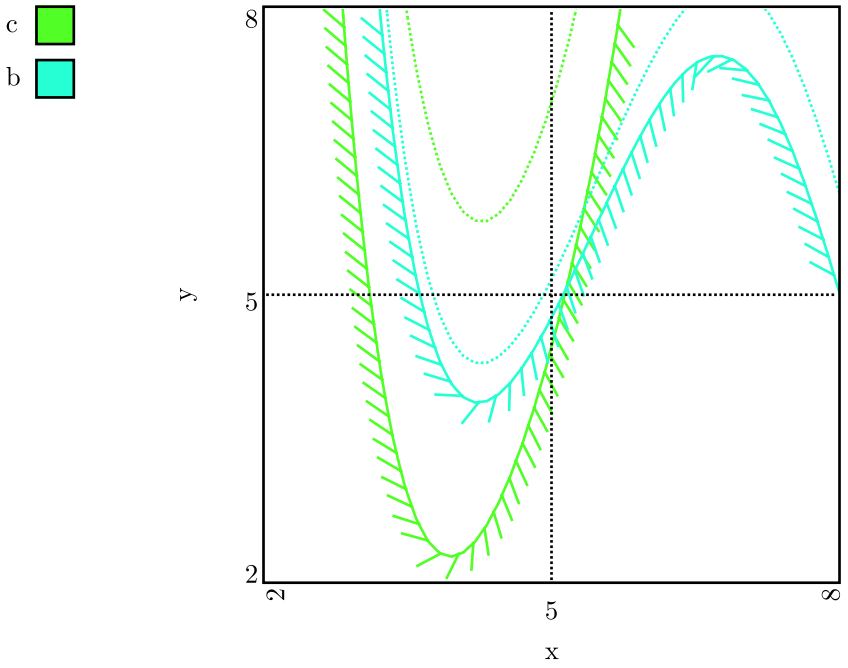

Figure 24: Constraint diagram

8.3.2 Change of Hairlines

To illustrate the behavior of this system and the system visualization to a change in point of interest, the hairlines were moved to the point listed in Table 5. The error and constraint levels were not changed.

Table 5: Perturbed input

Variable	Value
x	7.0
y	6.0
z	3.0

The change in point of interest lead to a corresponding change in system response as listed in Table 6.

The system propagated error at the altered point of interest is broken down in Figure 25.

Table 6: Perturbed response

Variable	Value
a	0.923
b	0.311
c	1.693
d	2.810×10^{-3}

In this case, the percent error in d is very large because of the very small response value of 2.81×10^{-3} in d.

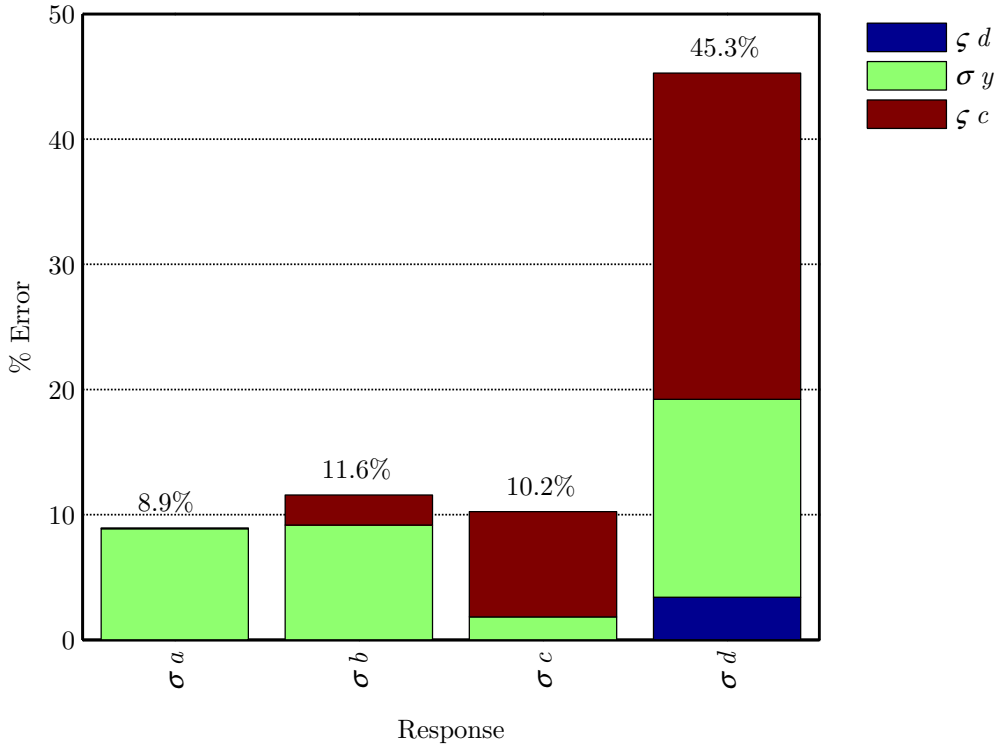

Figure 25: Error breakdown at alternate point of interest

The system response obtained by varying each input variable individually, but centered about the new point of interest, is presented in Figure 26.

The constraint diagram corresponding to the changed point of interest is included in Figure 27. The changes in x and y can be seen by the changed hairline locations on the chart. The change in z is demonstrated by the changes in the constraint and error curves

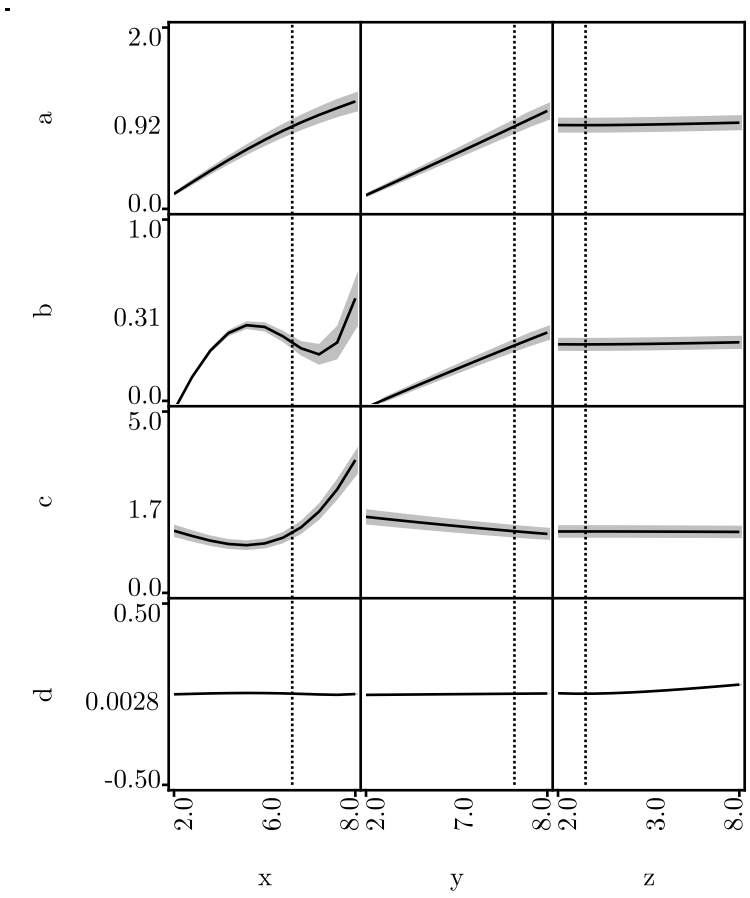

Figure 26: System response about alternate point of interest

themselves.

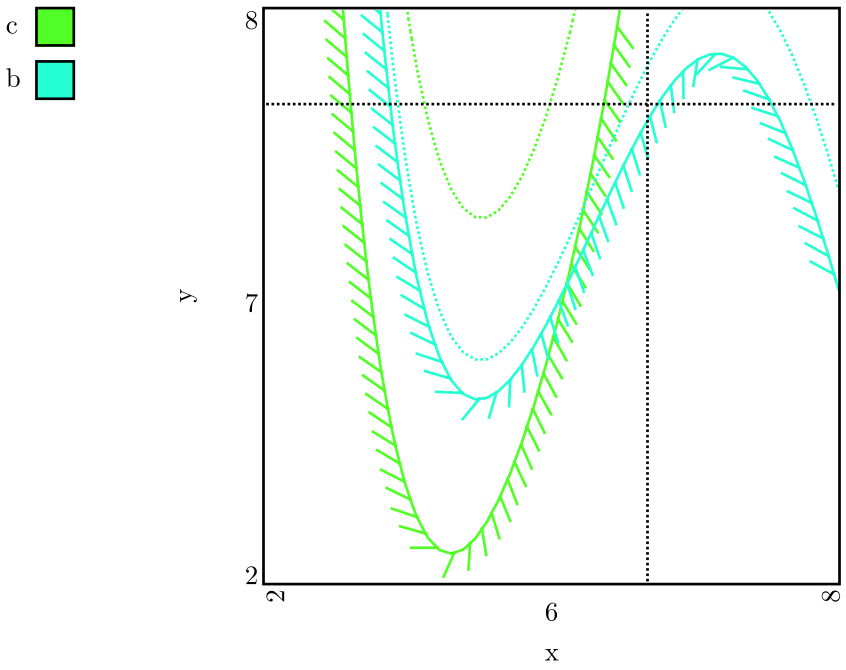

Figure 27: System constraints at alternate point of interest

8.3.3 Change of Error

To illustrate the behavior of this system and the system visualization to a change in error levels, the error levels were adjusted to the levels listed in Table 7. The point of interest and constraint levels were set to the baseline values.

Table 7: Perturbed error

Error Source	Value
ςd	0.075
σy	0.1
ςc	0.025

The system propagated error at the baseline point of interest, but for the adjusted error levels is broken down in Figure 28.

The system response for the adjusted error levels is presented in Figure 29. Note that the system response is identical to the baseline response shown in Figure 23, but that the

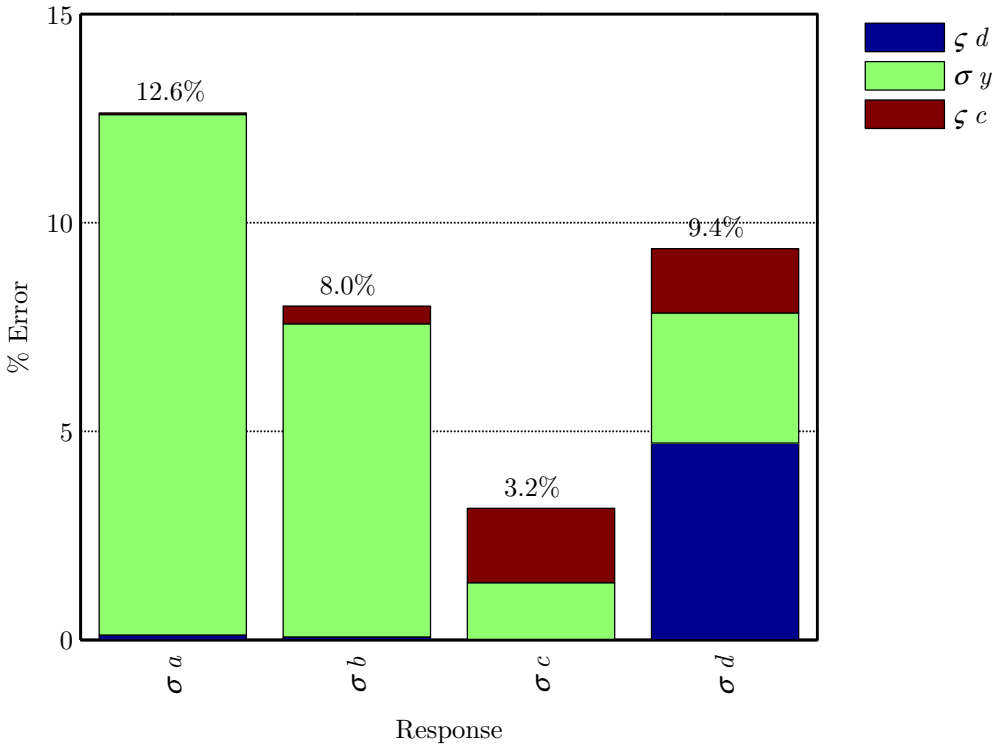

Figure 28: Error breakdown with adjusted error levels

error band has changed.

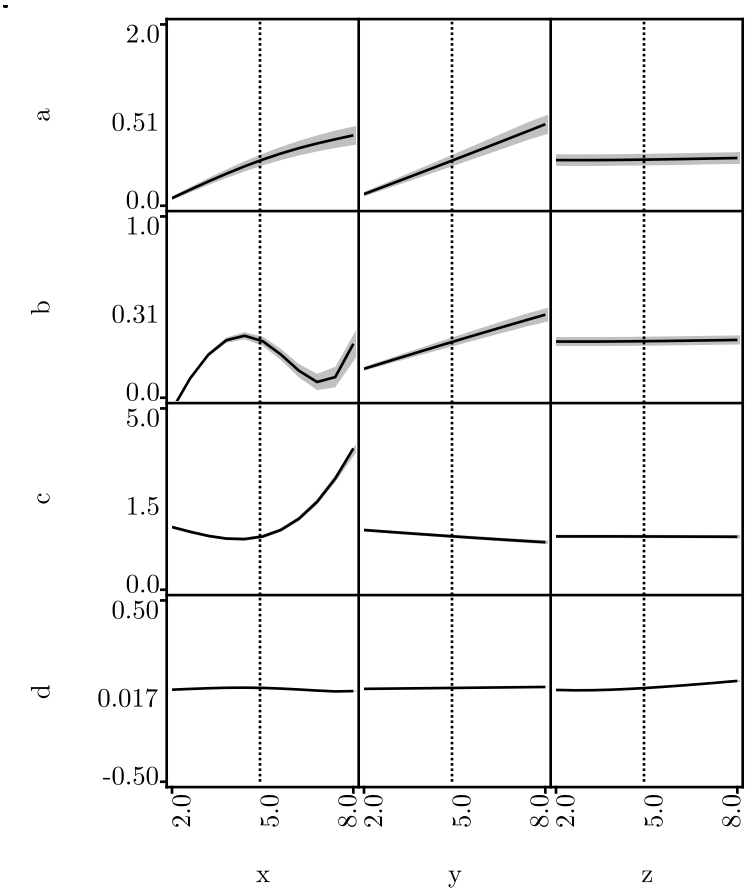

Figure 29: System response with adjusted error levels

The constraint diagram corresponding to the baseline point of interest, with adjusted error levels was included in Figure 30. Note that the constraint locations are identical to the baseline constraints shown in Figure 24, but that the error band has changed.

8.3.4 Change of Constraint

To illustrate the behavior of this system and the system visualization to a change in constraints, the constraint levels were adjusted to the levels listed in Table 8. The point of interest and error levels were set to the baseline values.

The point system response, error breakdown, and system response throughout the design space are unchanged by a change in the constraints. The constraint diagram corresponding to the baseline point of interest, with adjusted constraint levels, was included in Figure 31.

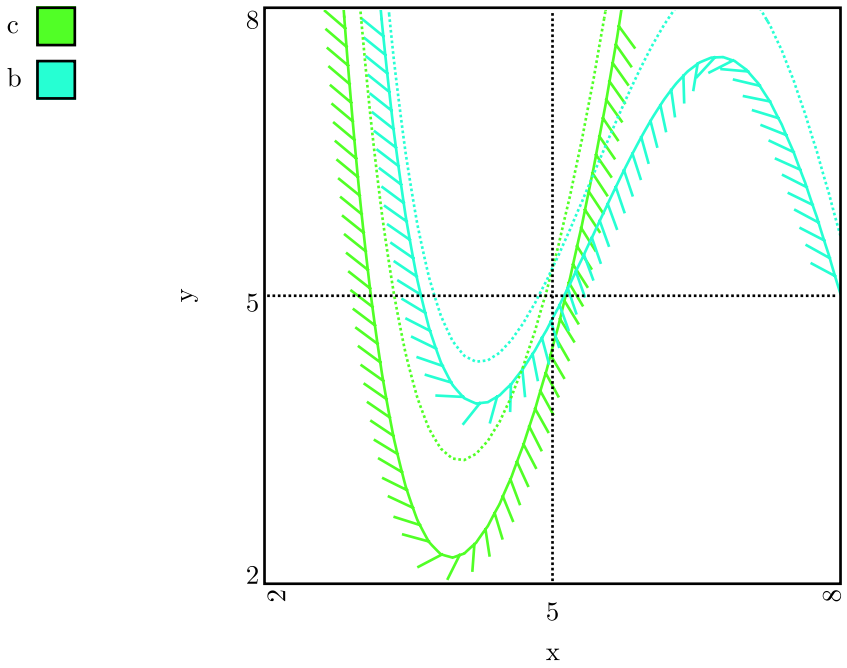

Figure 30: System constraints with adjusted error levels

Table 8: Alternate constraints

Constraint
$c > 1.32$
$b > 0.2$

Both the constraint locations and the locations of the constraints with error considered have changed from the baseline settings depicted in Figure 24.

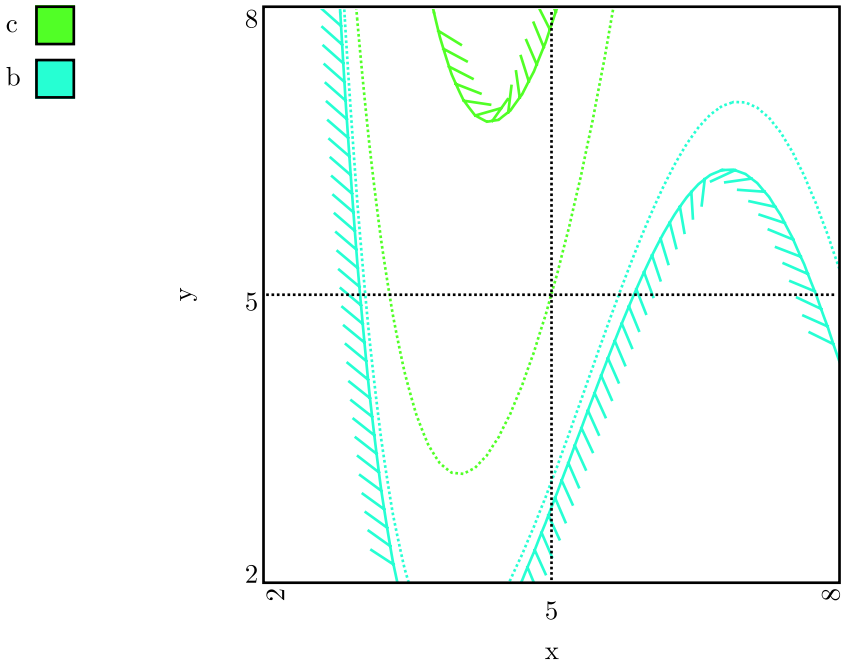

Figure 31: System constraints with adjusted constraint levels

CHAPTER IX

GAUSSIAN PROCESS METAMODEL COMPONENT

In the statistics community, the term Gaussian process refers to a specific type of stochastic process; a stochastic process is a random function. In the metamodeling community, the term Gaussian process refers to a metamodeling technique that relies on Gaussian random functions. In this work, a Gaussian process in the statistical sense will be called a Gaussian random function.

9.1 Gaussian Process Metamodel Theory

The following discussion of the theoretical foundation of Gaussian processes is intended as a basic explanation to aid in understanding their behavior and limitations; non-trivial steps are omitted. It is not meant as a rigorous proof or derivation. The literature contains numerous treatments of Gaussian process metamodels including Gibbs and MacKay [37], Williams and Rasmussen [64], Koehler and Owen [33], Neal [65], MacKay [36], and MacKay [66] which were used extensively in this research. During the execution of this research, a new comprehensive text [67] on Gaussian processes for regression and classification has been published, and should be used as a primary resource for future work. Any of these references may be consulted for clarification of theoretical points not fully explained by this treatment.

9.1.1 Random Functions

A random function maps from a domain of possible functions to a given function in the same way a random variable maps from domain of possible values to a given value. Just as the domain of a random variable may or may not encompass all numbers, the domain of a random function may or may not encompass all functions. Take for example, the following simple one-dimensional functional form, where the coefficients b_i are independent normally distributed random variables with zero mean and variance σ_i^2. This pedagogical discussion

of random functions follows closely that given by Santner et al. [68].

$$y(x) = b_0 + b_1 x + b_2 x^2$$

This functional form, when combined with the statement describing the distribution of the coefficients, defines a domain of functions from which a function may be drawn at random. The domain of this random function is all quadratic functions. In addition to defining the domain from which a function may be drawn, a random function defines the probability of each occurrence. In this random function, large coefficients are unlikely while near zero coefficients are most likely.

As with a random variable, we can calculate the expected value (mean) of $y(x)$. Note that this is the expected value of y at a particular (albeit variable) x, not the expected value over a random variable or a range of x.

$$E\{y(x)\} = E\left\{b_0 + b_1 x + b_2 x^2\right\}$$

Noting that the expected value of a sum is equal to the sum of the expected value of the terms, and recognizing that x is not a random variable, gives the following form.

$$E\{y(x)\} = E\{b_0\} + E\{b_1\} x + E\{b_2\} x^2$$

Of course, $E\{b_i\} = 0$, and so $E\{y(x)\} = 0$ for any given x. This agrees with intuition, for any term is equally likely to be positive as it is negative. We can also calculate the variance of $y(x)$.

$$Var\{y(x)\} = E\left\{[y(x) - E\{y(x)\}]^2\right\}$$

Using the fact that $E\{y(x)\} = 0$ and manipulations similar to those used above, we find that.

$$Var\{y(x)\} = \sigma_0^2 + \sigma_1^2 x^2 + \sigma_2^2 x^4$$

Similarly, we can calculate the covariance between the function response at two input points, x_1 and x_2.

$$Cov\{y(x_1), y(x_2)\} = E\{[y(x_1) - E\{y(x_1)\}][y(x_2) - E\{y(x_2)\}]\}$$

Which can be shown to be

$$Cov\left\{y\left(x_1\right), y\left(x_2\right)\right\} = \sigma_0^2 + \sigma_1^2 x_1 x_2 + \sigma_2^2 x_1^2 x_2^2$$

This quantity is defined as the covariance function $C\left(x_1, x_2\right) \equiv Cov\left\{y\left(x_1\right), y\left(x_2\right)\right\}$ for a random function. The covariance function is a statement about the relationship (similarity or difference) between two responses $y\left(x_1\right)$ and $y\left(x_2\right)$ sampled from the random function, based on the relationship between x_1 and x_2.

The covariance function defines the domain of a random function and the probability of each possible function within that domain. The domain of the random function is defined through attributes such as continuity, smoothness, limiting behavior, and periodicity. The covariance function does not specify any functional form of the random function.

9.1.2 Gaussian Random Functions

A Gaussian random function is a specific kind of random function $y\left(\mathbf{x}\right)$ where, for any number $(0 < n < \infty)$ of points $\left(\mathbf{x}_1, \ldots, \mathbf{x}_n\right)$ the joint distribution of random variables $\left(y\left(\mathbf{x}_1\right), \ldots, y\left(\mathbf{x}_n\right)\right)$ is jointly Gaussian.

As a Gaussian random variable is completely specified by its mean and variance, a Gaussian random function is completely specified by its mean function and covariance function. As is common in the Gaussian process community, we consider only Gaussian random functions with a zero mean function.

A necessary and sufficient condition for a random function to be Gaussian is that the covariance function be positive semidefinite.

9.1.3 Bayesian Inference

Bayes' theorem may be thought of as a quantification of Occam's Razor, which can be stated as a "preference for the simplest of models which agree with observation." Preference is quantified by the use of probabilities. Each model considered presents a hypothesis H. The observations combine to form the evidence E.

$$P\left(H|E\right) = \frac{P\left(E|H\right) P\left(H\right)}{P\left(E\right)}$$

Each of these terms have specific names outlined below describing their role in Bayes' theorem. The *posterior* is the probability of the hypothesis given the evidence $P(H|E)$. The *likelihood* is the probability of the evidence given the hypothesis $P(E|H)$. The *prior* is the probability of the hypothesis $P(H)$. The *evidence* is the probability of the evidence $P(E)$.

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

Metamodeling may be viewed as the act of finding an approximation of the generating function corresponding to a set of data, although in some cases, the key activity is to perform prediction based on the approximate function and the function itself need not be known. The approximate function obtained is $y(\mathbf{x})$. The corresponding set of data is a series of function observances y_i at input points $\mathbf{x}_i$. The data set is made up of n observances. The observed responses are collected into a vector, $\mathbf{y}_n \equiv (y_1, \ldots, y_n)$ and the corresponding input vectors are collected into a matrix $\mathbf{X}_n \equiv (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

In Gaussian process metamodeling, Bayesian inference is used twice to come up with the best model. One layer of inference is to select the most likely Gaussian random function, while the next layer is to choose the most likely generating function from the Gaussian random function. The generating function is never explicitly obtained, but it may be used for prediction. The first level of inference will be discussed later in this chapter.

In the context of the second level of inference, we seek to confirm the hypothesis, the function $y(\mathbf{x})$, given the evidence (training data), $\mathbf{X}_n, \mathbf{y}_n$. The Gaussian random function provides the prior over the function space.

$$P(y(\mathbf{x})|\mathbf{X}_n, \mathbf{y}_n) = \frac{P(\mathbf{X}_n, \mathbf{y}_n|y(\mathbf{x}))\, P(y(\mathbf{x}))}{P(\mathbf{X}_n, \mathbf{y}_n)}$$

9.1.4 Prediction

Prediction is the process of producing an estimate of the response, y_{n+1}, at a test point, $\mathbf{x}_{n+1}$. We define $\mathbf{C}_n$ to be the matrix formed by evaluating the covariance function between the n training points. The covariance between the test point and the training points forms

the vector $\mathbf{k}$, while the auto-covariance of the test point is c. These combine to define $\mathbf{C}_{n+1}$ as follows.

$$\mathbf{C}_{n+1} = \begin{bmatrix} \mathbf{C}_n & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & c \end{bmatrix}$$

With these definitions, the Gaussian process prediction of a response and its variance are as follows.

$$y_{n+1} = \mathbf{k}^{\mathrm{T}} \mathbf{C}_n^{-1} \mathbf{y}_n$$

$$\sigma^2_{y_{n+1}} = c - \mathbf{k}^{\mathrm{T}} \mathbf{C}_n^{-1} \mathbf{k}$$

The Gaussian process prediction of the derivative vector is as follows. A study was performed to verify the derivative equation by comparing the analytical derivative to a finite difference derivative. The results of this verification study were included later in this chapter.

$$\left.\frac{\partial y}{\partial x_k}\right|_{n+1} = \frac{\partial \mathbf{k}}{\partial x_k} \mathbf{C}_n^{-1} \mathbf{y}_n$$

Recall that the prediction is based on a Gaussian random function, so our answer is in probabilistic form. The response, y_{n+1}, is the mean of the random function at the input point $\mathbf{x}_{n+1}$ and $\sigma^2_{y_{n+1}}$ is the variance of the random function at the point.

Note that prediction requires the inversion of $\mathbf{C}_n$ while $\mathbf{C}_{n+1}$ never appears. Therefore, the inverse of $\mathbf{C}_n$ may be computed once and used repeatedly for predictions.

Also note that while the evaluation of the response is a linear matrix operation, the response is not limited to linear behavior. A Gaussian process metamodel is not a linear model; it is a linear smoother [69]. A smoother acts to smooth the observed $\mathbf{y}_n$ points to come up with a prediction [70]; the behavior of the training points is not limited in any way. The test point coordinates $\mathbf{x}_{n+1}$ appear in the prediction only through the covariance vector $\mathbf{k}$. The Gaussian process metamodel uses the relationship of the input coordinates to make a statement about the relationship of the response. Relationship is judged through similarity.

This is also the reason that a Gaussian process metamodel has no global functional form. Other examples of metamodels with no functional form include linear interpolation and cubic splines. Linear interpolation uses the relationship between a test point and the surrounding training points to make an approximation for the response at that point. In this way, similarity is used to make a prediction without a functional form. These examples have locally defined functional behavior, but there is no global functional form. With sufficient and appropriate training data, these models can approximate, to any desired level of accuracy, over an arbitrarily large domain, any function, be it exponential, polynomial, logarithmic, periodic, discontinuous, etc.

The behavior of a Gaussian process metamodel is entirely determined by the covariance function and the training data. The covariance function quantifies our prior knowledge of the function. If we know the function has a linear trend, is continuous, smooth, periodic, or has a discontinuity, we choose a form of the covariance function that supports that knowledge. The covariance function determines the domain of the function space that we draw from. It also defines the probability of drawing each function within the domain.

9.1.5 Covariance Function

Together the covariance function and the assumed zero mean function completely describe the Gaussian random function used in the Gaussian process metamodel. A random function is defined in essentially the same way as a random variable; a Gaussian random variable is described by its mean and its variance. The definition of a random variable determines the domain of the random variable (as a subdomain of all numbers) and the probabilities of each possible value in the domain. Likewise, the definition of a random function determines the domain of the random function (as a subdomain of all functions) and the probabilities of each possible function in the domain. The covariance function allows one to build knowledge of the underlying function into a model. However, this comes at the risk of making a poor choice; if a function does not appear in the domain, there is no way it will appear in the model.

Recall that the covariance function $C(\mathbf{x}_1, \mathbf{x}_2)$ is the covariance between the *responses*

at a pair of input points $\mathbf{x}_1$ and $\mathbf{x}_2$. A necessary and sufficient condition for a function to be the covariance of a valid Gaussian random function is that it must be a positive semidefinite function. The construction of valid positive semidefinite functions is non-trivial. Fortunately, much work has been done in this field, and catalogues of covariance functions with interesting behavior may be found in the literature [71].

Covariance functions may be classified by how the pair of input points $(\mathbf{x}_1, \mathbf{x}_2)$ appear in the function. If a covariance function is only a function of the (signed) distance between the input pair, $(\mathbf{x}_1 - \mathbf{x}_2)$, it is invariant with translations of the input space, and is called stationary. If the covariance function is only a function of the magnitude of the distance between the input pair, $(|\mathbf{x}_1 - \mathbf{x}_2|)$, it is invariant to rigid motions of the input space, and is called isotropic. If the covariance function is only a function of the dot product of the input pair, $(|\mathbf{x}_1 \cdot \mathbf{x}_2|)$, it is invariant to rotations of the input space about the origin, and is called a dot product covariance function. In general, covariance functions are classified as stationary or non-stationary.

The covariance function used in this work is given below as Equation 14; it is a form of the covariance function used most frequently in the literature. The covariance function is isotropic (and therefore also stationary). The covariance function is parameterized in terms of a set of hyperparameters $\mathbf{\Theta} \equiv (\theta_1, \theta_3, r_1, \ldots, r_m)$, for a function of m input coordinates. This covariance equation corresponds to a domain of functions that are infinitely differentiable, and thereby very smooth. These functions can be shown to correspond to an infinite number of radial basis functions.

$$C(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{\theta_1} \exp\left[-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^2 \cdot \mathbf{r}\right] + \delta_{ij}\theta_3 \tag{14}$$

The m derivatives needed for the prediction of response derivatives are given below, where $\mathbf{x}^*$ represents the point that "moves" in the derivative, and $\mathbf{x}$ represents the stationary point being compared against. In practice, $\mathbf{x}^*$ will correspond to the test point while $\mathbf{x}$ will correspond to each of the training points in building the covariance derivative vector $\frac{\partial \mathbf{k}}{\partial x_k}$.

$$\frac{\partial C(\mathbf{x}, \mathbf{x}^*)}{\partial x_k^*} = C(\mathbf{x}, \mathbf{x}^*)\, r_k\, (x_k - x_k^*)$$

The θ_1 hyperparameter controls the overall scale of variation of the function. The θ_3 hyperparameter controls the scale of input independent noise. The $(r_1, \ldots, r_k)$ hyperparameters are assembled into a vector $\mathbf{r}$ and act as a measure of the length scale of variation in each of the m input directions. Note that the definition of the θ_1 and $\mathbf{r}$ hyperparameters are inverted from that usually found in the literature. This was done to improve the numerical behavior of the optimizer used later in this process, and does not impact the formulation in any other way. Of course any interpretation of the meaning of the hyperparameters must be adjusted accordingly.

Other covariance functions are possible and should be applied when sufficient knowledge about the problem dictate the need for their behavior. These include forms with less smoothness, including step changes, as well as periodic behavior and spatially varying length scales suitable for modeling exponential behavior. Furthermore, these functions may be combined into more complex functions while maintaining their positive semidefinite character. Addition, multiplication, and convolution are among the operations available to compose covariance functions.

9.1.6 Choosing Hyperparameters

The covariance function presented in the previous section is parameterized to encompass a family of covariance functions. Any valid choice of hyperparameters corresponds to a valid Gaussian random function with its own domain and probability distribution. The act of prediction uses Bayesian inference to choose the best function from a Gaussian random function given the training data. Any set of hyperparameters corresponds to a Gaussian random function, and may be used for prediction; however, they will not produce equally good predictions. We must next choose the best set of hyperparameters to be used for prediction.

Recall that in Gaussian process metamodeling, Bayesian inference is used twice to come up with the best model. One layer of inference is used to select the most likely Gaussian random function, while the next layer is used to choose the most likely generating function from the Gaussian random function.

In the context of the first level of inference, we seek to confirm the hypothesis, the hyperparameters $\boldsymbol{\Theta}$, given the evidence, $\mathbf{X}_n, \mathbf{y}_n, y(\mathbf{x})$. Our prior assumptions about the behavior of the hyperparameters appear as the hyper-prior, $P(\boldsymbol{\Theta}|y(\mathbf{x}))$. Bayes' theorem for this layer of inference is given below.

$$P(\boldsymbol{\Theta}|\mathbf{X}_n, \mathbf{y}_n, y(\mathbf{x})) = \frac{P(\mathbf{y}_n|\mathbf{X}_n, y(\mathbf{x}), \boldsymbol{\Theta}) P(\boldsymbol{\Theta}|y(\mathbf{x}))}{P(\mathbf{y}_n|\mathbf{X}_n, y(\mathbf{x}))}$$

The use of Bayes' theorem at this stage requires the evaluation of an intractable integral over all possible hyperparameters. Two approaches to this problem are customary in the field. The first involves a Monte Carlo approximation of the integral. The other approach uses an optimization process to maximize the likelihood of the data in terms of the hyperparameters. The latter method is the approach adopted in this work.

By maximizing the likelihood of the observed data given the Gaussian random function (by varying the hyperparameters), we find the Gaussian random function that best explains the data. The second level of inference (discussed previously) chooses the best function for prediction from the Gaussian random function. For improved numerical behavior, the optimizer equivalently operates to minimize the negative log likelihood instead of operating to maximize the likelihood. The objective function and its derivative are depicted below.

$$-\ln P(\mathbf{y}_n|\mathbf{X}_n, y(\mathbf{x}), \boldsymbol{\Theta}) = \frac{1}{2}\ln\det\mathbf{C}_n + \frac{1}{2}\mathbf{y}_n^{\mathrm{T}}\mathbf{C}_n^{-1}\mathbf{y}_n + \frac{n}{2}\ln 2\pi$$

$$\frac{\partial}{\partial\Theta}\left[-\ln P(\mathbf{y}_n|\mathbf{X}_n, y(\mathbf{x}), \boldsymbol{\Theta})\right] = \frac{1}{2}\mathrm{tr}\left(\mathbf{C}_n^{-1}\frac{\partial\mathbf{C}_n}{\partial\Theta}\right) - \frac{1}{2}\mathbf{y}_n^{\mathrm{T}}\mathbf{C}_n^{-1}\frac{\partial\mathbf{C}_n}{\partial\Theta}\mathbf{C}_n^{-1}\mathbf{y}_n$$

A lognormal hyper-prior is applied to give the user a degree of control over the search properties of the optimizer. The hyper-prior effectively establishes a statistical set of move limits for the optimizer; the optimizer may search locally, within a certain number of orders of magnitude. The hyper-prior makes large changes in the point of interest unlikely.

The lognormal distribution was chosen for the hyper-prior because it produces only positive values and to facilitate interpretation of the distribution parameters [72]. When a lognormal probability distribution is plotted on a logarithmic scale, it appears to be a normal distribution. The distribution is positioned such that the optimizer's initial guess for the hyperparameter is the most likely result. The width of the distribution is set by the

standard deviation of the normal distribution on a logarithmic scale; the width is set by specifying a number of orders of magnitude. This sets the size of the move limits.

While the use of a hyper-prior has been proposed in the literature, the author could find no references where a hyper-prior was actually used to train a Gaussian process metamodel. The use of a lognormal hyper-prior to provide an intuitive means for the user to control the hyperparameter optimization process represents a unique contribution of this research.

The lognormal distribution for a random variable x with μ and σ equal to the mean and standard deviation of the random variable's logarithm is given below.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \frac{1}{x} \exp \frac{-(\ln x - \mu)^2}{2\sigma^2}$$

The derivative of the lognormal distribution is as shown below,

$$\frac{\partial P(x)}{\partial x} = P(x) \left(\frac{-\ln x + \mu - \sigma^2}{\sigma^2 x} \right)$$

and the derivative of the log probability, which is used in the objective function, is given below.

$$\frac{\partial}{\partial x} \ln P(x) = \frac{-\ln x + \mu - \sigma^2}{\sigma^2 x}$$

The width parameter is set as follows by the user specifying the number of orders of magnitude Ω for the standard deviation.

$$\sigma = \ln 10^{\Omega}$$

The mean parameter is set as follows such that the mode of the distribution is equal to the optimizer initial guess x_0.

$$\mu = \sigma^2 + \ln x_0$$

The conditional probability of an independent event is equal to the probability of the event, so $P(\boldsymbol{\Theta}|y(\mathbf{x}))$ simplifies to $P(\boldsymbol{\Theta})$ if the hyper-prior is assumed independent of the Gaussian random function. Similarly, the joint probability of a set of independent hyperparameters is equal to the product of the probabilities of each hyperparameter. This rule extends trivially to the log probability and probability derivatives discussed above.

The introduction of the optimization problem to choose the hyperparameters also introduces the danger of choosing a locally optimum set of hyperparameters rather than the global optimum. Conceptually, this corresponds to choosing a wrong length scale or noise level for the Gaussian random function. For example, given a few noisy data points from an otherwise well behaved system, one may choose to fit a high frequency (short length scale) model which passes through the points (small noise level), or one may choose to use a low frequency (long length scale) model which only passes near the points (large noise level). The hyper-prior is an ideal mechanism which may be used to incorporate prior knowledge to suppress erroneous solutions. In this case, a hyper-prior that favored smooth, longer length scale models would eliminate the errant local minimum.

In the context of using Gaussian process metamodels to model deterministic computer experiments, there is no noise in the measurement. This well behaved situation is not expected to exhibit problems with local minima. In fact, the noise model could be omitted entirely, but it is thought the extra degree of freedom introduced by the noise model is useful to allow for the very small noise levels associated with numerical studies.

Furthermore, in practice the user will see that a poor set of hyperparameters has been chosen and should have some intuition on how to supply a better initial guess to allow the optimizer to find the desired solution. The hyper-prior will help to keep the optimizer in the region of the suggested solution.

Another step was taken to improve the numerical behavior of the Gaussian process metamodel and the optimizer. All quantities were normalized to be on a similar scale and range. The exact normalization procedure is not critical so long as it results in quantities of the same order of magnitude. In this study, input and output quantities were scaled and shifted such that they fell approximately within the range of zero to one.

The gradient based optimizer actually operated on the log of the hyperparameters. This this is standard practice for likelihood maximization and greatly improved the ability to find good optima. This makes sense in light of the nature of hyperparameters; it seems that while their magnitude is very important and can vary widely, the exact value of the hyperparameter is not so important. This transformation was wrapped around the optimizer

itself such that the rest of the Gaussian process metamodel and the data repository did not have to be modified for the transformation. Furthermore, in this manner, other transformations could be experimented with quickly. The transformation for the hyperparameters and the gradient of the objective function are given below.

$$x = \ln \theta$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial \theta}\theta$$

The JMinuit optimizer from the FreeHep Java library [73] was used in this research. This optimizer is not capable of enforcing arbitrary constraints; however, it does have the capability to apply bounds constraints on the design variables. This feature was used to make sure all the hyperparameters remained larger than 1×10^{-6}. Experience showed that hyperparameters smaller than this value lead to poor behavior of the optimizer and the Gaussian process metamodel; this behavior was most likely caused by the rounding error introduced by the finite precision storage of the training points in the data repository.

9.1.7 Training Data

The training data used in Gaussian process metamodeling is of obvious importance. The more training data one has, the better one's model will be. However, the size of the covariance matrix grows with the number of training points. The covariance matrix must be inverted at each step in the optimization process. Depending on the number of training points and the dimensionality of the optimization problem, the matrix inversion may be prohibitively expensive in the training procedure.

In this research, the training data is limited to function value observances at points. Some Gaussian process metamodels are also able to make use of observances of function derivatives at points. If analytical derivatives are produced by the program being approximated at little cost, then the incorporation of derivative information can significantly improve the quality of the metamodel.

This metamodel has been integrated with a data repository with the intent of expanding to contain a very large data set corresponding to all past studies. There may be situations

in which the user does not want to use the entire data set in the metamodeling activity, so there should be some straightforward controls for limiting the data set. Two approaches to data set limiting are immediately obvious: a spatial limit and a chronological limit. Spatial limiting would allow the user to restrict the data used to the region of interest. Chronological limiting would allow the user to observe the historical state of the data set and thereby the metamodel.

Various techniques have been explored in the Gaussian process metamodeling community to mitigate the matrix inversion cost with a large training set [37]. These techniques include covariance functions leading to sparse matrices and optimization using only a subset of the training points [74]. For applications using Gaussian processes to model deterministic computer codes, the matrices should not grow excessively large for modern desktop computers. Therefore, these techniques have not been investigated in detail. However, should they become necessary, promising techniques exist in the literature and should apply directly.

9.2 Gaussian Process Metamodel Interface

The Gaussian process metamodel described above has been implemented in a program with a graphical user interface. This interface is first used for developing and tuning a metamodel. Then the interface may be used for exploring the metamodel. This section highlights the primary features of the interface. Some additional features and the detailed behavior of the system interface are discussed in Appendix C.

The metamodel can be accessed in multiple ways. The metamodel can run in stand-alone mode, with the interactive GUI presented for tuning the model, and it can also run as an Analysis Server component, responding to requests for an estimate at a point. While running as an Analysis Server component, the metamodel can run with the interactive interface active or disabled. With the interface active, the user may tune the metamodel while running a study using a client like ModelCenter. Once the metamodel is well established, it can be locked down to assure its behavior does not change. At that point, it would be appropriate to run the metamodel in the batch mode with the GUI disabled. Also, if performance demands warrant, the metamodel may be directly integrated into another

application such as the system interface, which is described later. Source level integration eliminates any overhead introduced by the Analysis Server protocol.

The metamodel interface provides for controlling all aspects of the metamodel behavior as well as a mechanism appropriate for facilitating decision making about how the metamodel should be changed based on its behavior.

A depiction of the metamodel interface immediately after startup is included in Figure 32. The function being modeled is a function of two variables $c = c(x, a)$. There are thirty data points in the database, and all of them have been retrieved. The hyperparameters are initialized to reasonable default values. The width of the hyperprior is set to 2.0 for two orders of magnitude. The normalization factors are all set to have no effect: zero shift and unitary scaling.

The user interface depicts the complete state of the metamodel. In the upper left corner, there is a radio button controlling whether the metamodel is in frozen or dynamic mode. In frozen mode, the training data set is fixed. In dynamic mode, the interface polls the database every ten seconds to see if more training points have been added. Below the radio button, there is another panel for interacting with the training data. In dynamic mode, the panel reports the number of points in the data set, and the number of points added the last time the database was polled. There is a refresh button which will force the program to poll the database for new points immediately. In frozen mode, the index state field becomes active. This allows the user to constrain the data points to those with an index value less than a certain limit. The index value is incremented as each point is queued, so this allows chronological control of the data set. In frozen mode, the refresh button polls the database for points within the constraint. As expected, the total number of points are displayed and the number of points added reflects the state as of the most recent action.

Below the training data control box, there is a panel for interacting with the hyperparameters. At the top of the panel there is a box for controlling the width of the hyper-prior. Below that, there is a spreadsheet displaying all of the hyperparameters. In this case, there was only one response and two input variables. The spreadsheet has one column for each response and one row for each hyperparameter. There are two base hyperparameters and

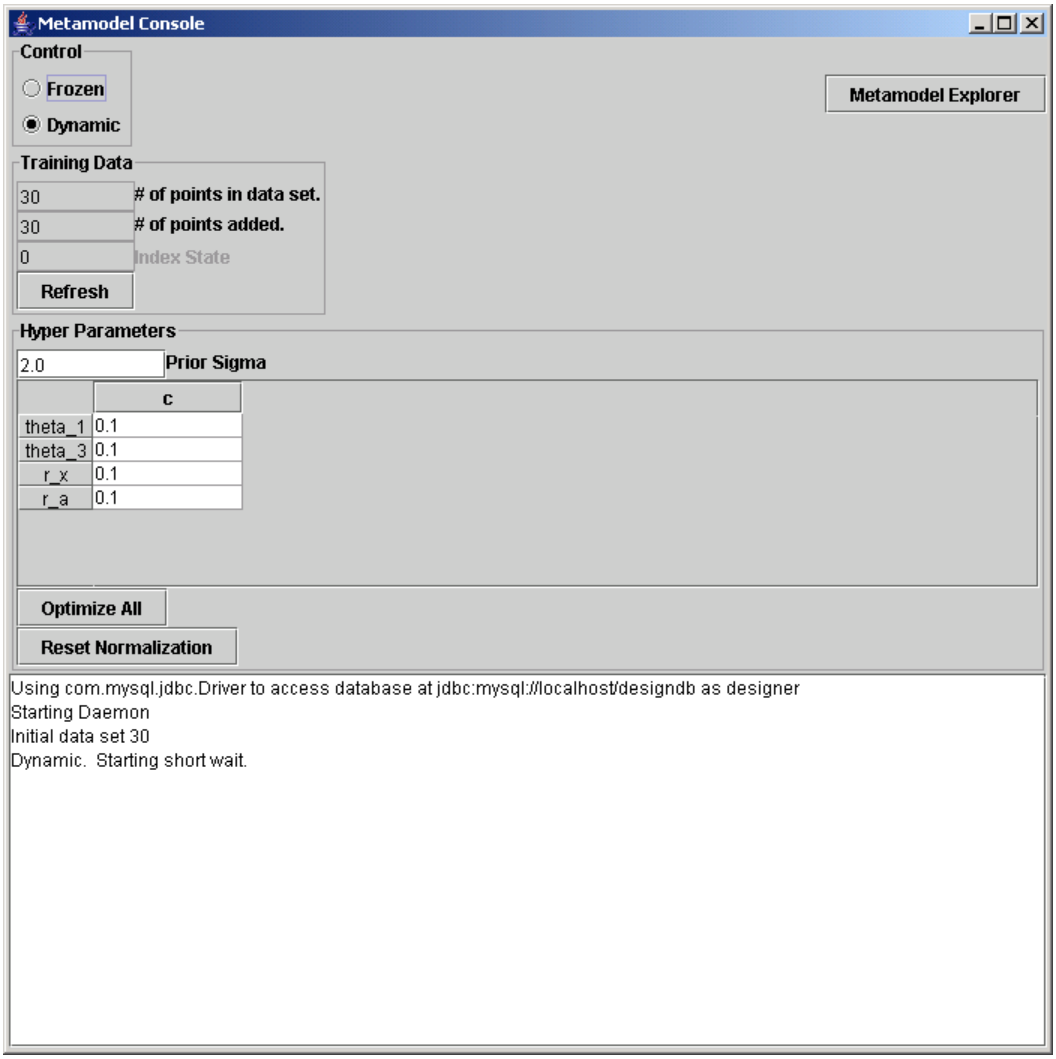

Figure 32: Metamodel control panel immediately after launch

one for each input direction. If the spreadsheet grows beyond the size of the panel, scroll bars appear to allow the user to navigate the entire spreadsheet. The data scrolls, but the row and column headers remain visible.

Clicking on each column header will cause the metamodel interface to optimize the hyperparameters for that response. The current hyperparameters are used as the initial guess. Below the spreadsheet panel, there is a button that will cause the interface to optimize the hyperparameters for all of the responses.

Below the optimization button, there is a button that can be used to reset the normalization constants. When the button is pressed, a confirmation dialog, shown in Figure 33, appears reminding the user of the implications of resetting the normalization parameters. If the user chooses to continue, the program automatically sets the normalization parameters based on the range of the variables observed in the data set. Typically, the exact settings of the normalization parameters are not important and they only need to be set when the data set is initialized. If, however, in the process of performing a study, sufficient data points are added to change a representative value or the range of values of the data set by an order of magnitude, the normalization should be reset. This will cause the hyperparameters to lose their validity and they must be re-optimized.

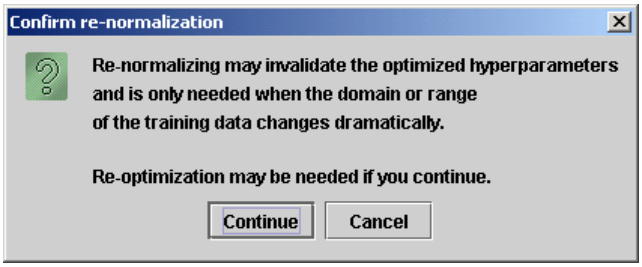

Figure 33: Metamodel normalization confirmation dialog

At the bottom of the interface, there is a text console used to report on the status of the metamodel. This is an output-only console, and as information is reported, the old information scrolls off the top of the display, at which point a scroll bar appears allowing the user to review old messages.

9.2.1 Metamodel Explorer

At the top right of the metamodel interface, there is a button which will launch a metamodel explorer interface. An example of the metamodel explorer interface corresponding to this initialized state is included as Figure 34.

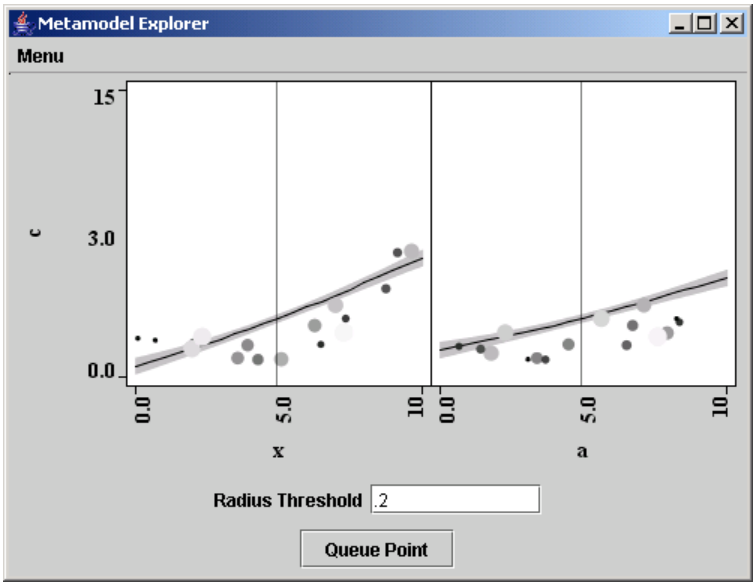

Figure 34: Initial metamodel explorer interface

The metamodel explorer presents an interface substantially similar to the system explorer discussed in Section 8.2.3. The primary differences lie in the fact that the system explorer represents the entire complex system as a function of the system level variables while the metamodel explorer represents a single component as a function of its local variables. In a similar vein, the gray band behind the response curve in the system explorer represents the system propagated error, while the corresponding band in the metamodel explorer represents the variance of the Gaussian random function used in the metamodel.

Furthermore, the response value throughout the input space is indicated by a line crossing each box in the grid. The line may be thought of as a slice through the response surface. It represents the function value obtained by varying that particular input (per the column) while holding all other inputs constant. It may be thought of as a variation in the same sense as a partial derivative.

Figure 35 included below depicts the slices through the domain presented by the metamodel explorer. The cutting plane of each slice is shown and curve where the plane cuts the function surface is highlighted. This three-dimensional surface plot is very effective for a function of two variables, but is not possible for higher-dimensional problems. The metamodel explorer extends effectively to problems of any dimensionality.

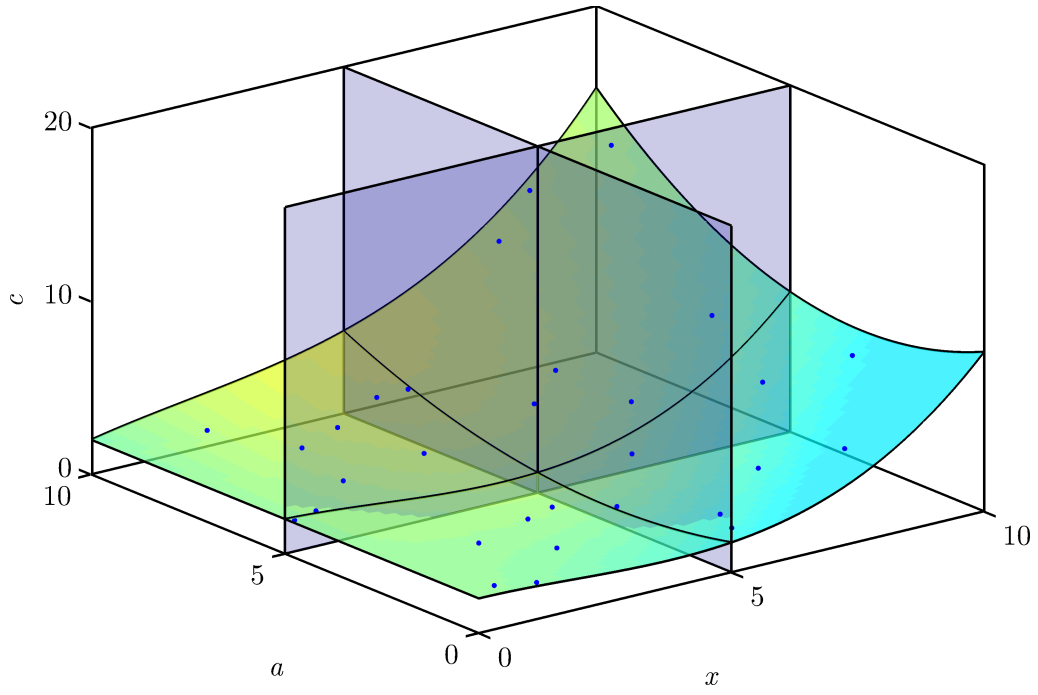

Figure 35: Metamodel explorer slices and training points

The gray band surrounding this line indicates the variance of the Gaussian process. The variance does not correspond directly to the error of the underlying metamodel. Loosely speaking, it represents how well the current Gaussian process (hyperparameter settings) agrees with the training data. A good metamodel will have a very thin (nearly nonexistent) band. However, if there is insufficient training data, a metamodel may nevertheless have a vanishingly thin band but still not do a good job of predicting the response throughout the domain. A thin variance band is necessary but not sufficient for a model to generalize well.

While this may seem nonintuitive (vanishing variance with poor predictive capability), it is a direct consequence of the Occam's Razor interpretation of Bayesian inference, i.e. choose the most simple model which agrees with the observances. In a one-dimensional

problem with only two training points, the Gaussian process will have a high degree of confidence in a model with linear behavior, even though other more complex models could explain the observances equally well. If the underlying function is actually more complex, say cubic, this linear model will be a poor predictor. However, if one increases the training set to include four data points, the Gaussian process will have a low degree of confidence in a model with linear behavior, as it will not agree with the observations. Furthermore, the Gaussian process will have a high degree of confidence in a model with cubic behavior, as it is the most simple model that agrees with the observations. The addition of more training points (which agree with the cubic model) further cements this confidence.

Finally, the metamodel explorer contains a depiction of the training points used to define the metamodel. In a one-dimensional problem, the concept of a slice through the function space becomes degenerate and plotting the training points is straightforward. However, in a multi-dimensional problem any arbitrary slice through the function space is unlikely to pass through any training points. This can be seen in Figure 35 which also depicts a set of training data throughout a two-dimensional domain.

Of course simply plotting all of the training points on each of the charts would present so much information that the trends indicated by the data would not be visible. Consequently, for multi-dimensional problems, the points displayed are selected based on how close they are to the slice through the input space. The displayed points may be interpreted as being those points particularly relevant to defining the metamodel in the neighborhood of the slice. Points far from the slice do not play a strong role in defining the metamodel near the slice, and therefore do not need to be represented.

This method of displaying a multi-dimensional metamodel in conjunction with the contributing training data represents an advance of this research in visualization for understanding any kind of metamodel. This intuitive display is simple to implement and should be incorporated into existing metamodeling methods and tools.

The radius threshold for selecting which points are displayed can be adjusted by the user through the entry field below the metamodel explorer. The radius threshold is a non-dimensional constraint that controls how close a point must be to a slice in order to be

displayed. Because a point may be close to a slice in one dimension, but not in others, the points displayed in each column may not appear in the other columns. The data points are represented by gray circles, the size and intensity of which varies with the distance from the slice. Points that are very close to the slice appear as small black dots while points that are further from the slice appear as larger gray circles. Points far from the slice, near the limit of the constraint, appear as large, nearly white circles.

This point scaling is consistent with selecting points important to the definition of the metamodel in the neighborhood of a slice. Points nearest the slice (black point) play a strong role in determining the exact behavior of the metamodel in that region. Points further from the slice (gray circle) play a vague role in determining the approximate behavior of the metamodel in that region. Points near the constraint limit (off-white circle) and beyond (white or invisible) play a very weak role in determining the rough behavior of the metamodel in that region.

The user may adjust the radius parameter to get a satisfactory display; this setting will depend on the number of data points in the training set and on the dimensionality and linearity of the problem. The parameter should be adjusted such that the data points depict the expected behavior of the underlying function. Too small a radius will not display sufficient points; too large a radius will include points from far away in the design space, confusing the source of the variation. Experience has shown that this technique for representing multi-dimensional training data produces good results for problems of moderate dimensionality (up to about six dimensions) and nonlinearity. While more work is needed to produce intuitive displays for high-dimensional problems, this visualization advance developed in this research is a dramatic improvement over the traditional solution limited to one-dimensional problems.

Returning to Figure 34, below the radius threshold control, there is a button that allows the user to *Queue Point*s to be added to the training set. When the button is pressed, the dialog depicted in Figure 36 appears with the current hairline settings in a series of entry fields. The user may adjust the point and then queue the specified point when satisfied. This allows the user to identify a region of the input space that has been inadequately

covered, due to insufficient points, or due to the local behavior of the function needing more points. This is another example of providing the user with a simple tuning parameter and the interface tools needed to make good decisions quickly. This man-in-the-loop adaptation capability is a key benefit of using a local metamodel.

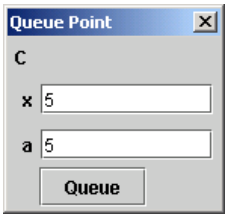

Figure 36: Queue point dialog

9.2.2 Typical Metamodel Use Case

The depiction of the metamodel interface given in Figure 32 shows what the user would see immediately after startup for a just-initialized metamodel. The input and response quantities have not been normalized and the hyperparameters have not been optimized. As can be seen from Figure 34, this metamodel is unacceptable. The metamodel does not go through or near the training data, especially the small black points which should be most significant to the model. There are some steps the user must take in order to obtain a high quality metamodel.

The metamodel interface is again depicted in Figure 37 after the user has switched to frozen mode and has normalized the inputs and responses. Observation of the output console reveals that the user left the metamodel in dynamic mode for a short time before switching to frozen mode. The messages printed during normalization can also be seen.

Once the quantities have been normalized, the user proceeds to optimizing the hyperparameters. This optimization may require some manual intervention and tweaking to obtain a satisfactory metamodel. The metamodel explorer should be consulted during the optimization progress to check the state of the metamodel. The metamodel inteface after a successful optimization process is depicted in Figure 38.

Examination of the output console reveals that the last actions by the user were repeated

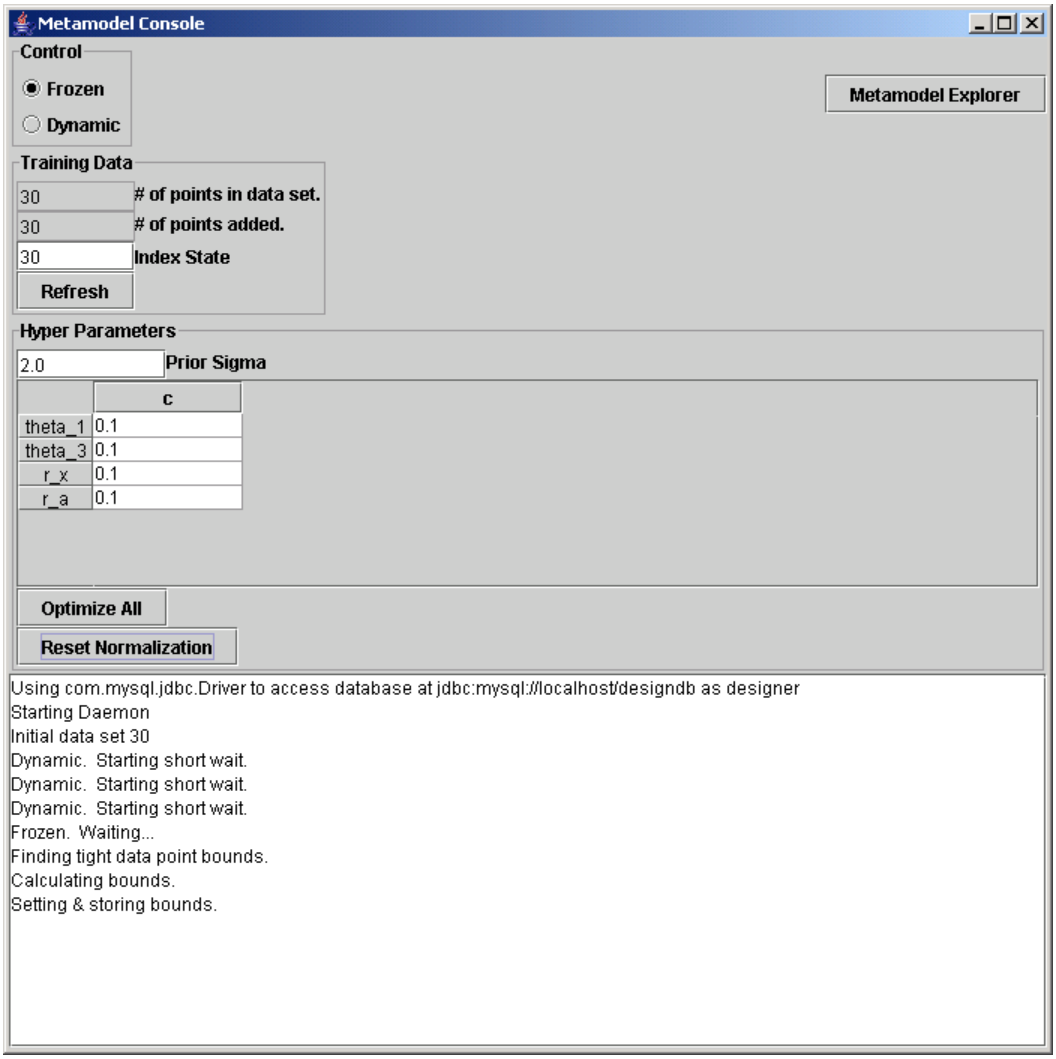

Figure 37: Metamodel control panel after normalization

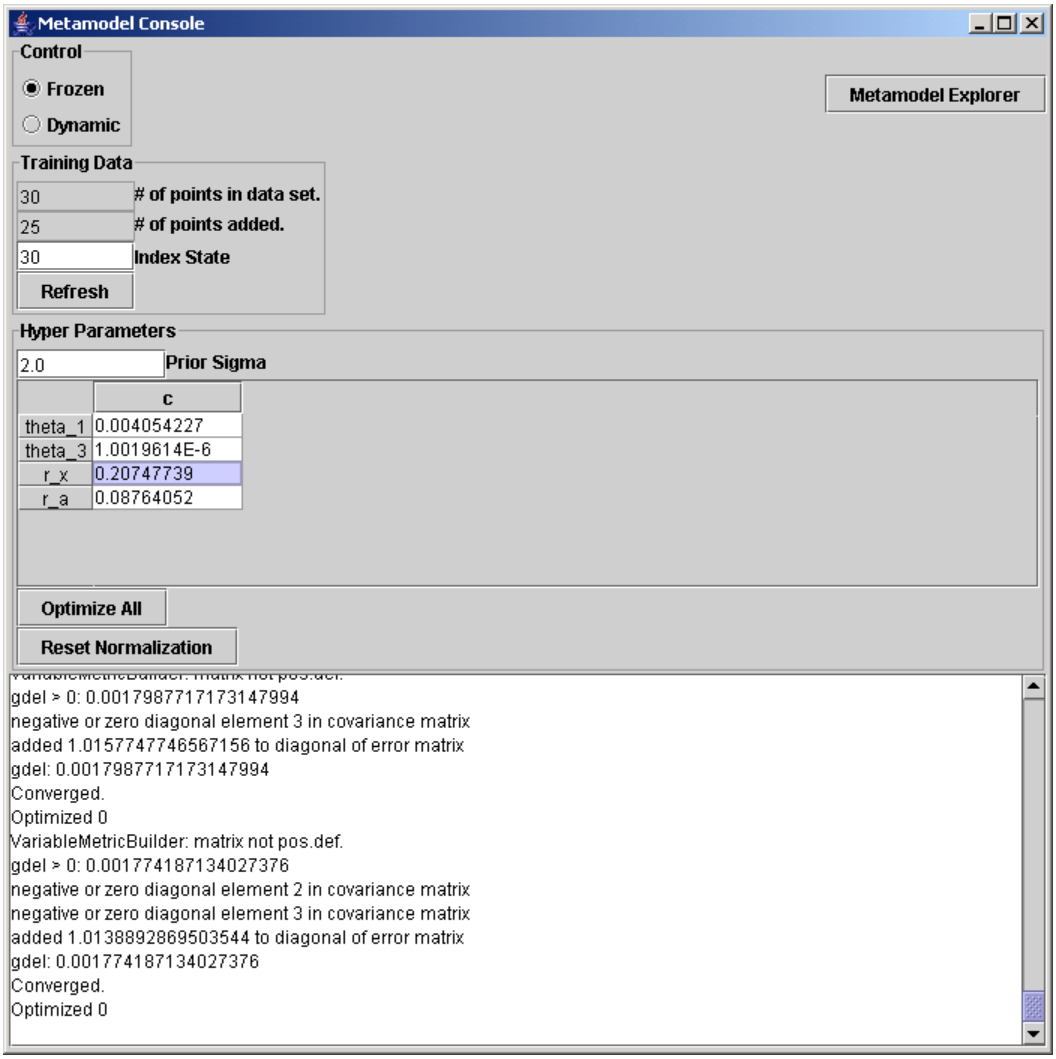

Figure 38: Metamodel control panel after optimization

runs of the gradient based optimizer. Both runs converged. The gradient based optimizer used in this research is rather verbose and outputs numerous diagnostic messages. It is good practice to re-run the optimizer once after an optimum has been found. Because the hyper-prior prefers the initial guess, this ensures that the hyper-prior has minimal impact on the final optimum selected.

9.3 Gaussian Process Metamodel Behavior

The behavior of a Gaussian process metamodel can be difficult to understand. However, some straightforward experiments can go a long way towards building some intuition regarding Gaussian processes and this user interface. To aid in this process, it is useful to keep in mind the exact function being approximated. In this case, the experiments are conducted on the function given below, this function was also displayed in Section 8.3 but is repeated here for convenience.

$$c = 2 + \left(x^2\left[x - 5.67834\right] + ax\left[a - 6.3432\right]\right)/60;$$

This function is cubic in x and quadratic in a. If the exact functional form were known and a least squares regression procedure was to be used to find the four undetermined coefficients and the intercept, only five appropriately distributed training points would be required.

For this experiment, an over-abundance of training data was created so the behavior of the metamodel could be observed with varying amounts of training data. The data set consisted of thirty points selected by a Latin Hypercube algorithm. The point were selected between zero and ten, with every value equally likely. Latin Hypercube sampling has domain spanning behavior; however, these properties of a Latin Hypercube design are lost when a subset of the data is used.

Figure 35 included earlier depicts a surface plot of the test function over the domain of interest, and has been repeated here for convenience as Figure 39. The points represent the thirty training points used for the Gaussian process metamodel. The two vertical planes represent slices through the domain represented in the metamodel explorer. The curve where the slices intersect the surface is highlighted.

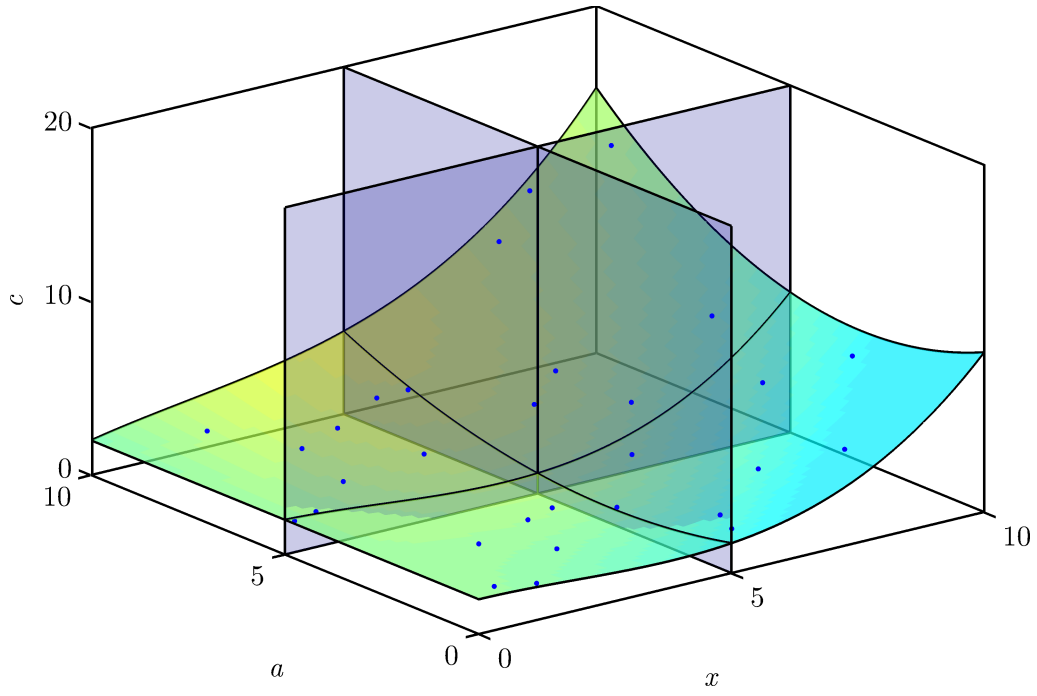

Figure 39: Test function with sample points

The hyperparameters were optimized as discussed in the previous section. The resulting hyperparameters have been included in Table 9; more significant figures than necessary have been included to allow for precise replication of the results. The corresponding metamodel is depicted in Figure 40. Note that the variance band is nonexistent, indicating a well-trained metamodel.

Table 9: Optimized hyperparameters

Hyperparameter	Value
θ_1	0.004054227
θ_3	1.0019614×10^{-6}
r_x	0.20747739
r_a	0.08764052

This metamodel was verified by comparing the metamodel response to the true response at 1000 random points. Three measures of metamodel quality were calculated, $\text{RMSE}_{\bar{x}}$ the root mean square error normalized by the mean, RMSE_{x_i} the root mean square relative error, and R^2 the square of the correlation coefficient. The equations for the error metrics

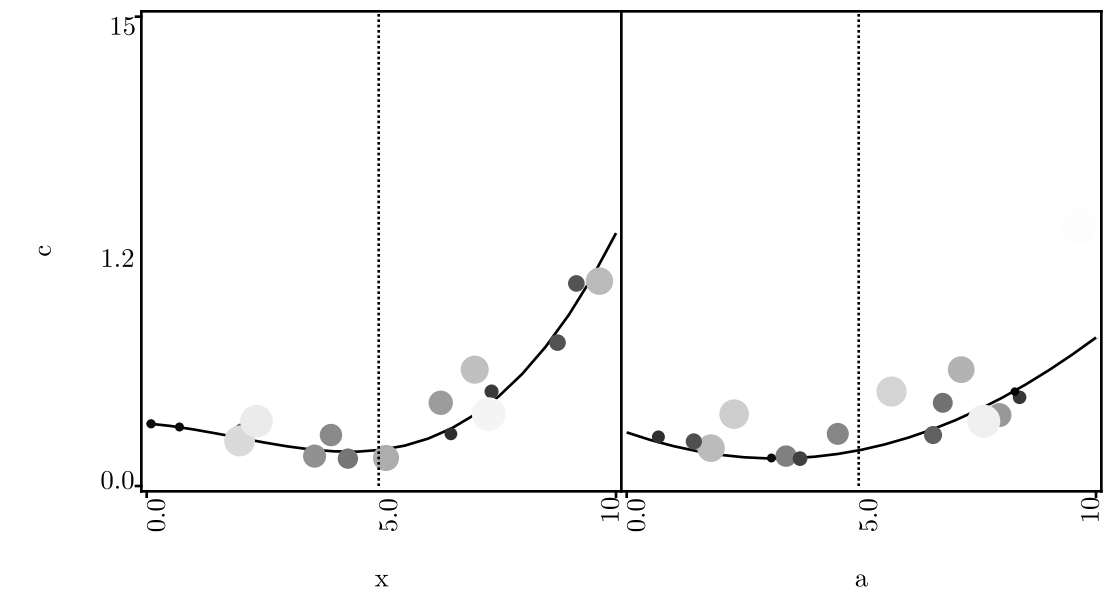

Figure 40: Metamodel with optimized hyperparameters

are given below and the three error metrics are listed in Table 10.

$$\text{RMSE}_{\bar{x}} \equiv \frac{1}{\bar{x}}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - x_i')^2}$$

$$\text{RMSE}_{x_i} \equiv \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i - x_i'}{x_i}\right)^2}$$

Table 10: Metamodel Characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
c	0.249%	0.207%	0.999993

All three of the metamodel quality metrics indicate a nearly perfect fit. As an additional test of the quality of the metamodel, the random observations of the actual response were plotted against the metamodel predicted response as Figure 41. In this figure, a perfect metamodel will have points which lie on a diagonal line while an imperfect metamodel will have points which fall in a cloud surrounding a diagonal line.

Figure 41 and the quality metrics listed in Table 10 indicate that the Gaussian process metamodel described above (the covariance function, training data, and hyperparameter

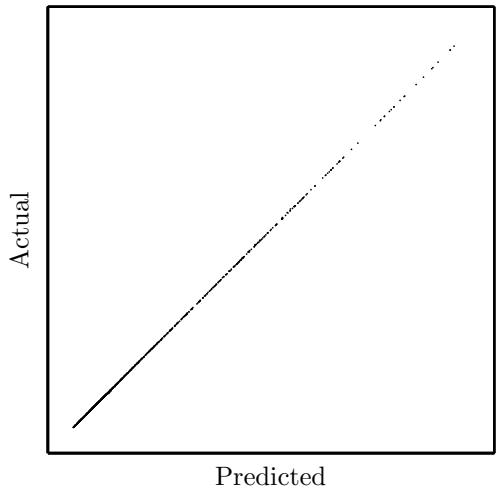

Figure 41: Metamodel verification

settings) provide an excellent approximation of the test function. This provides verification Gaussian process metamodel, the hyperparameter optimization, and their implementation.

9.3.1 Gaussian Process Derivative Verification

The equation presented in Section 9.1.4 for the analytical derivative of the Gaussian process metamodel was verified by comparing the analytical derivative of the Gaussian process and the finite difference derivative of the Gaussian process to the analytical derivative of the test function. The results for the partial derivatives $\frac{\partial c}{\partial x}$ and $\frac{\partial c}{\partial a}$ of the example test function are included as Figures 42 and 43 respectively.

The dotted horizontal line in Figures 42 and 43 represents the error in the analytical derivative of the Gaussian process relative to the true derivative of the test function. The small magnitude of this error level provides verification of the analytical derivative equation and of the metamodel itself.

The solid line in the figures represents the error in the first-order forward finite difference derivative of the Gaussian process relative to the true derivative of the test function. The magnitude of this error varies as the finite difference step size is changed. At the right, where the step size is large, there is much error due to the first order finite difference

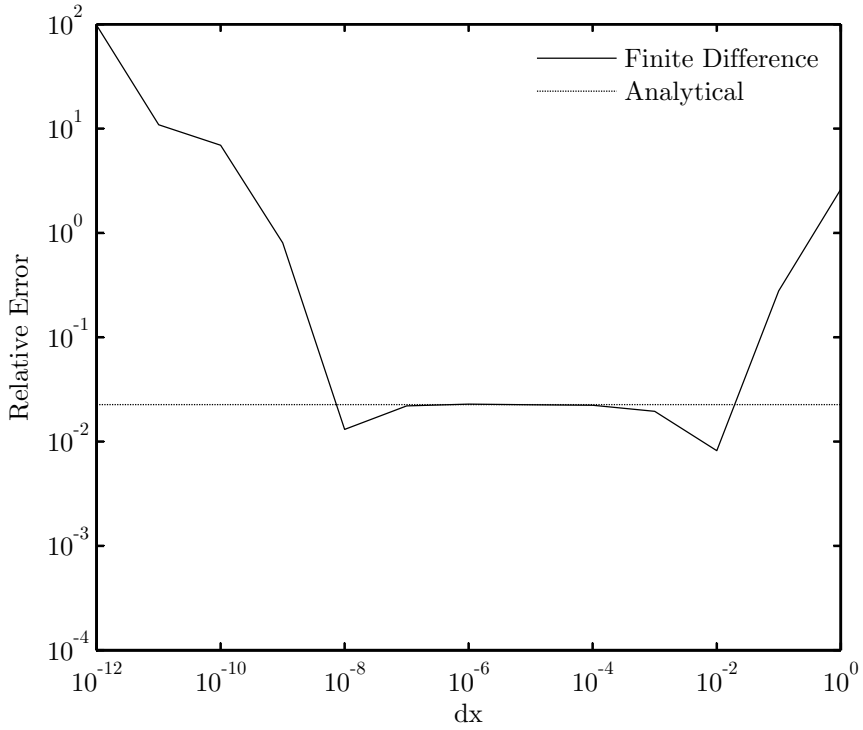

Figure 42: Analytical derivative verification $\frac{\partial c}{\partial x}$

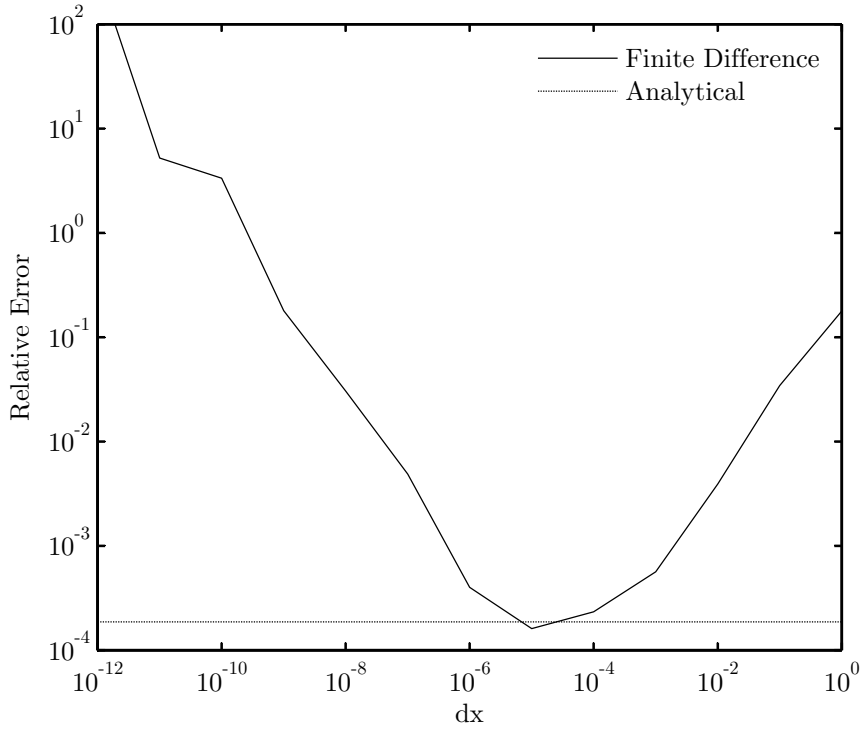

Figure 43: Analytical derivative verification $\frac{\partial c}{\partial a}$

approximation. At the left, where the step size is also large, there is much error due to floating point rounding caused by subtracting and dividing very small numbers. For intermediate step sizes, the finite difference derivative converges to the analytical derivative. This pattern of increasing error for increasing and decreasing step sizes is classical for finite difference derivatives. The convergence of the finite difference derivative to the analytical derivative provides verification of the analytical derivative equation.

9.3.2 Radius Threshold Variation

In Figure 40, about half of the training points are visible. Recall that the number of displayed points is controlled by the radius threshold parameter. In this case, the radius threshold was set to 0.2. As shown in Figure 44, when the threshold was increased to 0.5, all of the points became visible.

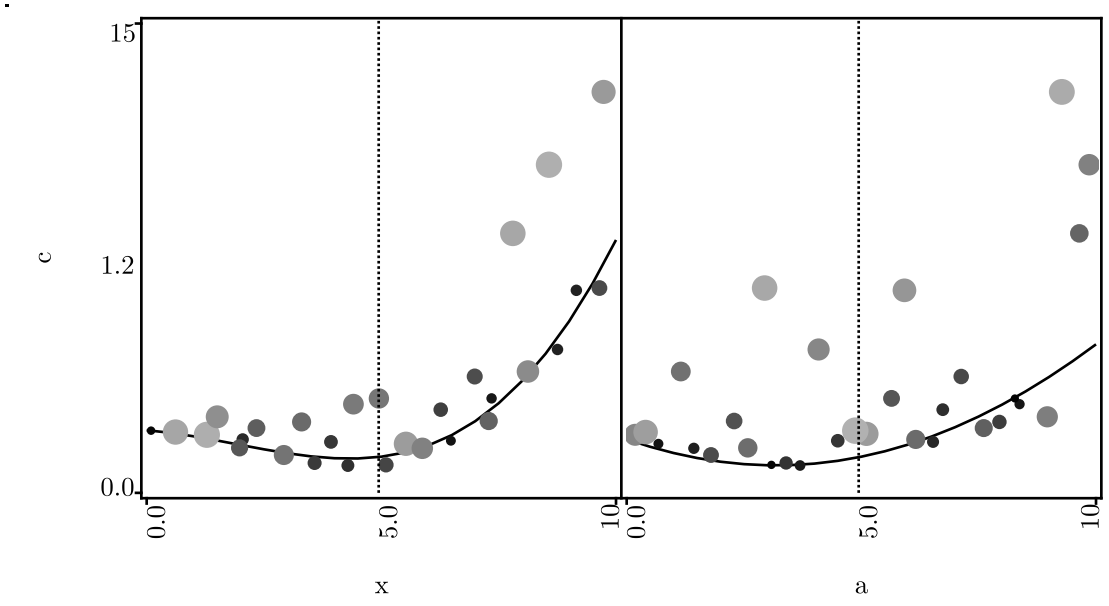

Figure 44: Increased threshold radius showing all training points

Similarly, as shown in Figure 45, when the threshold was decreased to 0.1, only the points very near the response were visible. This threshold setting seems to give good results, and will be the default in the following figures unless otherwise noted.

Inspection of Figures 40-45 demonstrates how an appropriate set of points relevant to the metamodel in the neighborhood of the slices can be selected and displayed in an intuitive

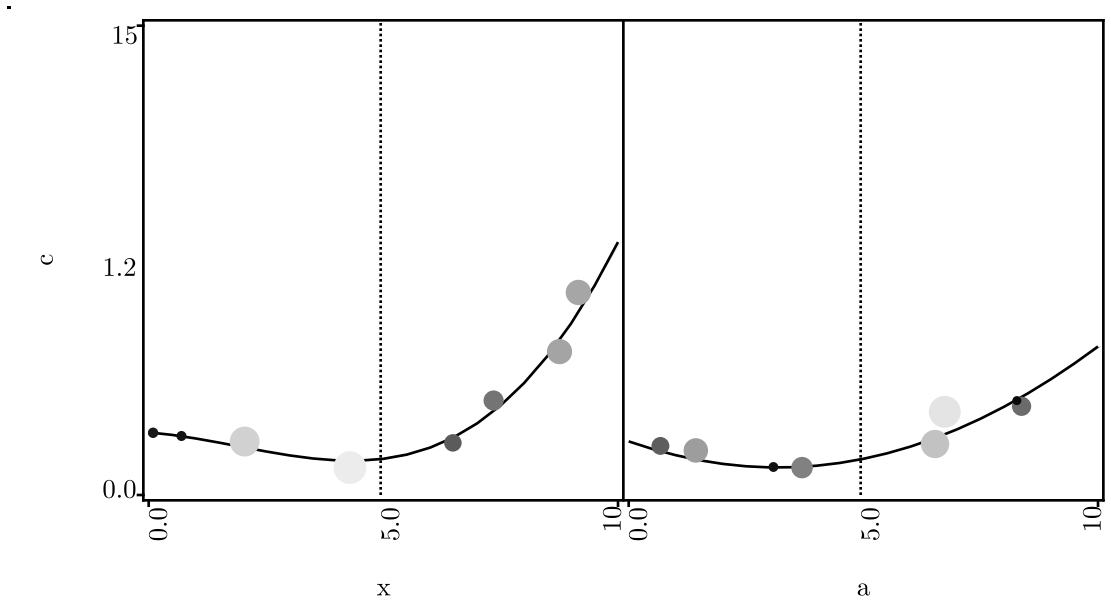

Figure 45: Reduced threshold radius highlighting the most significant training points

manner. Small dark points are very important to the behavior of the metamodel in that area while larger lighter circles are of diminishing importance. As expected, the small dark points fall nearest the function response curve, with the large lighter circles falling further from the response.

9.3.3 Hairline Variation

As described in the previous section, the user can change which slices through the design space are displayed by moving a hairline. Figures 46-49 depict a number of different slices through the design space. In each successive figure, only one hairline setting is changed. If the x direction is interpreted as increasing North, and the a direction is interpreted as increasing East, the regions explored are North, Northeast, Southeast, and Southwest respectively in the figures.

Figures 46-49 demonstrate how the metamodel explorer reacts to changes in the hairline settings. Note that the response curves move to represent a new slice through the design space. Note also that the quantified response value changes to reflect the new hairline setting. Of course, because the location of the slices changes, the training points displayed in each of the figures change; however, the intuitive behavior of only displaying the training

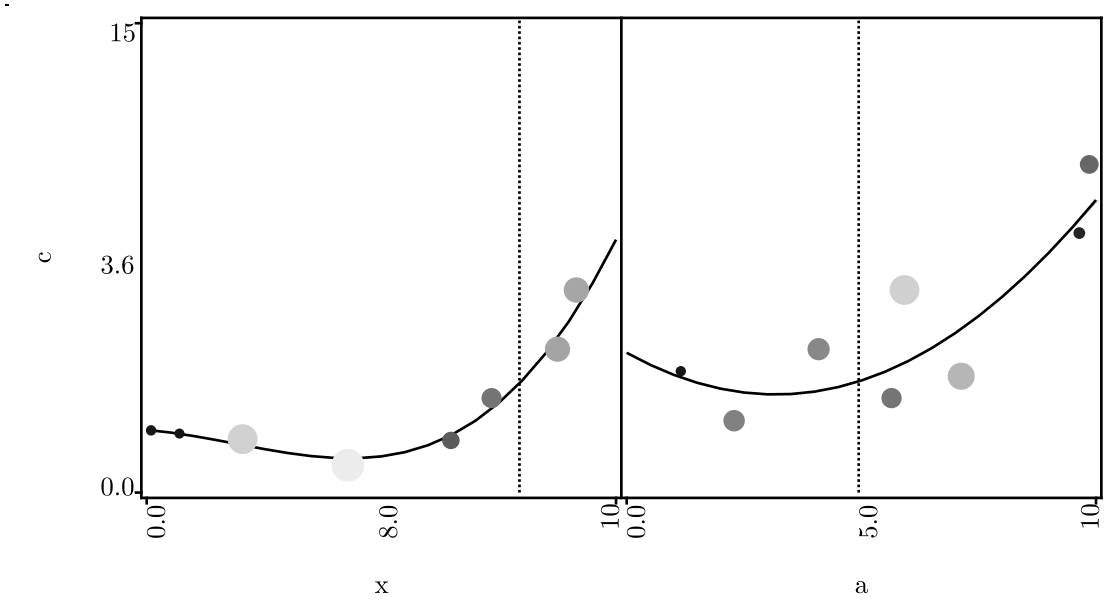

Figure 46: Hairlines set to northern region of design space

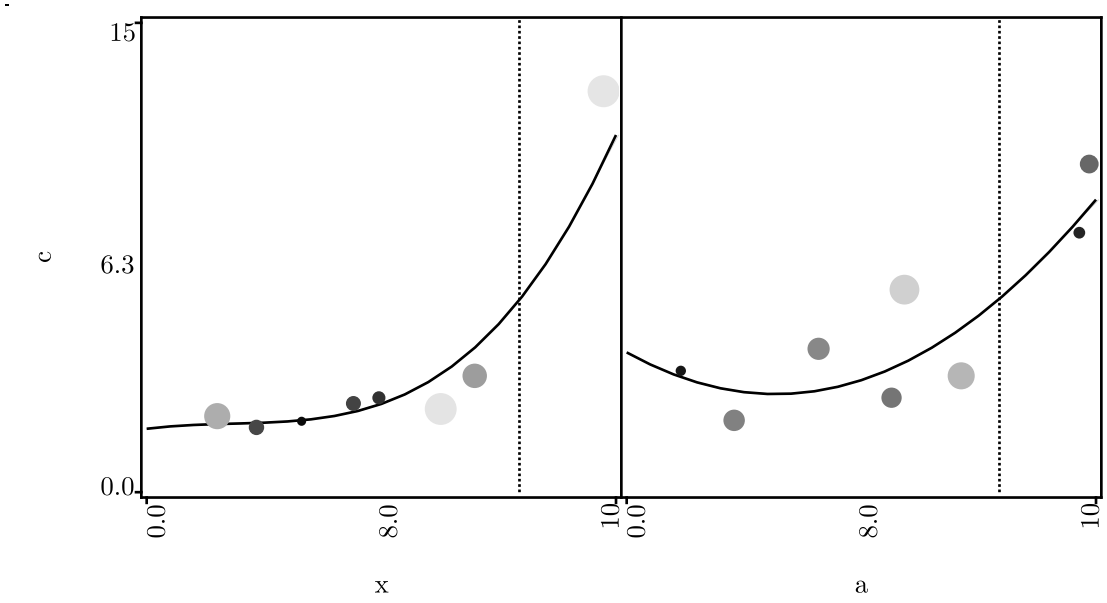

Figure 47: Hairlines set to northeast region of design space

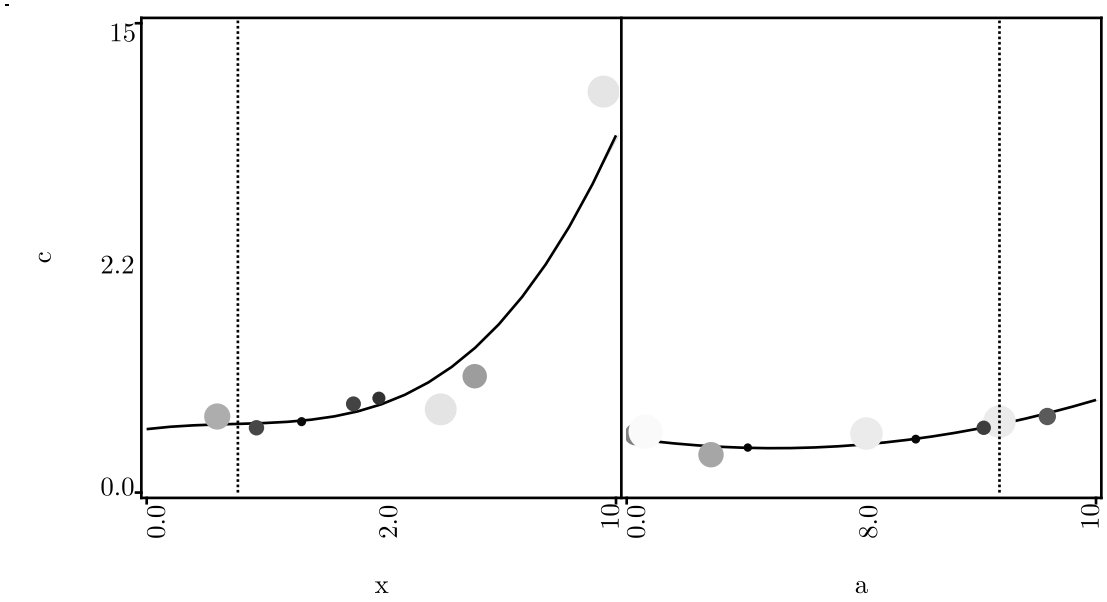

Figure 48: Hairlines set to southeast region of design space

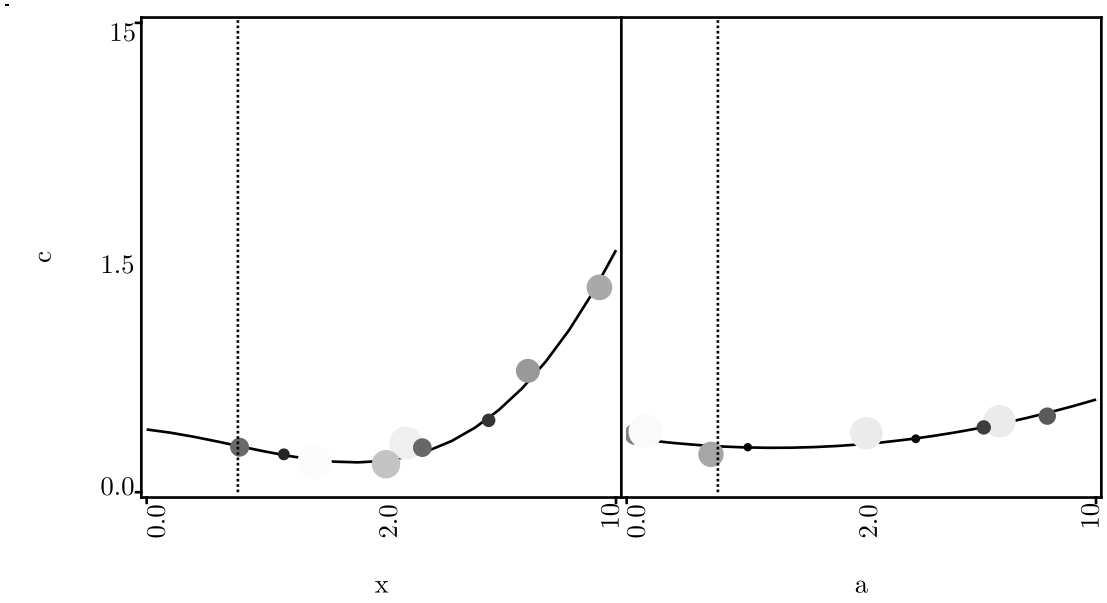

Figure 49: Hairlines set to southwest region of design space

points in the neighborhood of the slices is consistent. Finally, notice that when a single hairline is moved, that column in the metamodel explorer does not change; this is because of the partial derivative nature of the display.

The lack of a variance band throughout Figures 46-49 demonstrates that a high quality metamodel has been achieved throughout the domain. If, when exploring the domain with the hairlines, the user came across a region where the metamodel was poor, the user can select a point with the hairlines, and queue that point for execution. This new point will locally improve the metamodel. Depending on the amount of change introduced by the new training point, it may then be appropriate to re-optimize the hyperparameters.

9.3.4 Range Variation

Most metamodels are of minimal predictive utility when used to extrapolate beyond the training data. For example, the behavior of high-order response surfaces fit with least-squares regression can be particularly bad outside the training data. Furthermore, the extrapolation behavior of these models can change dramatically with a small change in model order or included training data.

The behavior of Gaussian process metamodels beyond the training data is typically far less erratic. In the case of the covariance function used in this research, far from the training data the response will smoothly return to the mean of the training data. The details of this behavior are situation dependent, but some common traits can be observed. Of course, it is not a good idea to use any metamodel far beyond the range of the training data.

To demonstrate its behavior beyond the range of the training data, Figure 50 depicts the optimized Gaussian process metamodel on a larger scale. The radius threshold has been decreased to 2×10^{-5} to suppress the display of any training points. Recall that the training data was chosen between zero and ten in both x and a, so this figure represents a tripling of the domain in each dimension.

Figure 50 clearly shows the expected cubic and quadratic behavior of the response throughout the domain. On this scale, the metamodel has been very well behaved, and could cautiously be used for prediction. The addition of a few more training points at

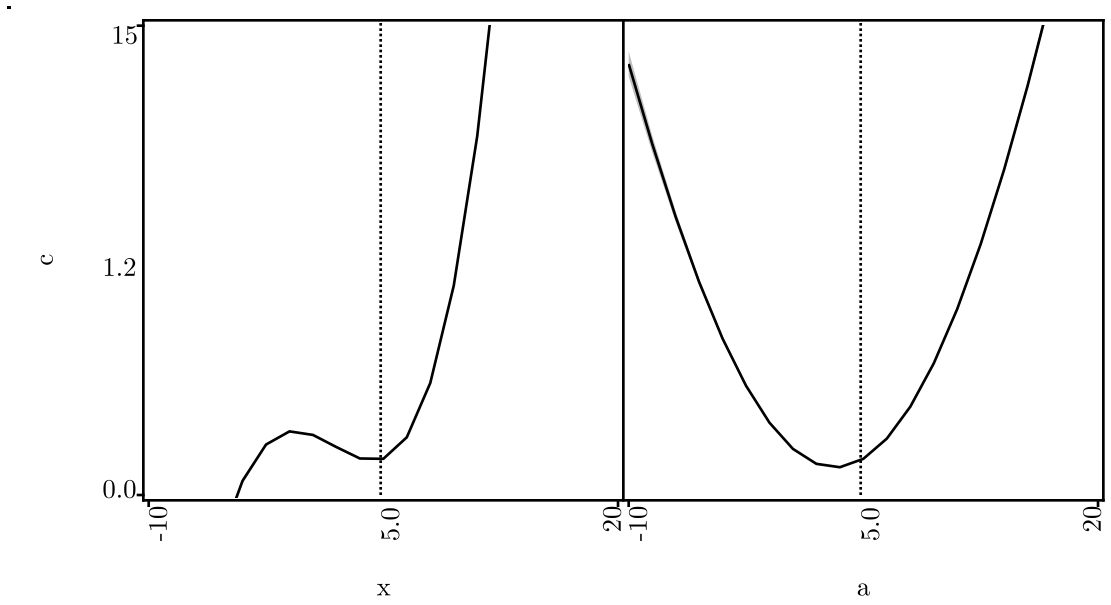

Figure 50: Metamodel viewed with tripled input ranges

the extremes of this expanded range could serve to pin down the metamodel if prediction throughout this domain were required.

Figure 51 further increases the scale of the metamodel to be displayed over a domain eleven times the domain of the training data in each dimension. The covariance function chosen for this research has a built-in assumption that a function is constant. Consequently, with no evidence to the contrary (outside influence of the training data), the Gaussian process metamodel returns to a constant value. This transition happens gently at the same time as the predicted variance band expands.

The constant-seeking property of this covariance function contributes to the relatively benign behavior of Gaussian processes when extended beyond the ranges of the training data. Other covariance functions may be used as required with inherent linear, periodic, or other behavior; Gaussian process metamodels based on these covariance functions will behave accordingly differently beyond the training data.

Figure 52 increases the scale of the metamodel to an extreme, increasing the displayed domain to be forty times that of the training data in each dimension. The non-smooth behavior of the response on this scale is due to the finite sampling resolution used in the

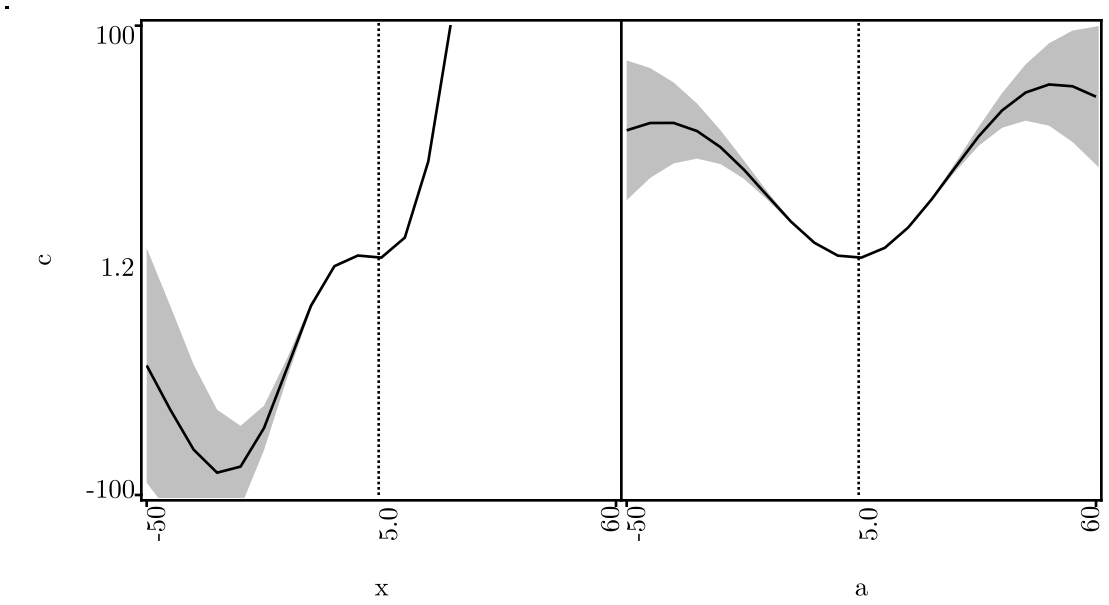

Figure 51: Metamodel viewed with very large ranges

generation of the plots and does not indicate non-smooth behavior of the Gaussian process. The return of the response prediction to a constant value can be clearly seen.

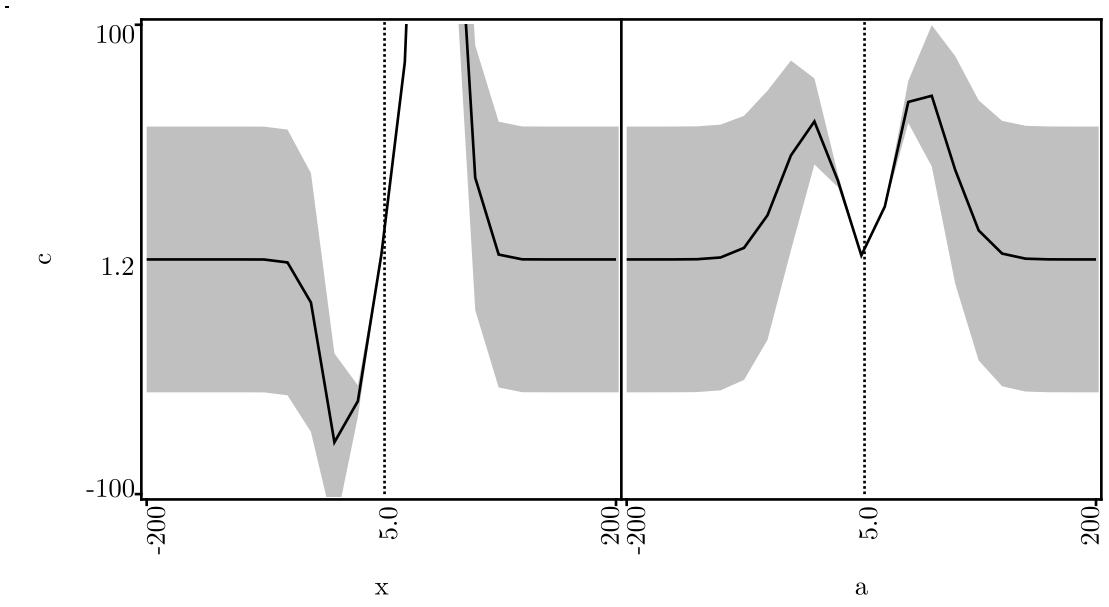

Figure 52: Metamodel viewed with extreme ranges

The addition of data points near the extremes of these ranges would do much to correct the behavior of the metamodel. However, doing so may make it necessary to re-normalize

the training data with the according re-optimization of the hyperparameters.

9.3.5 Hyperparameter Variation

In order to investigate and understand the impact the hyperparameters have on the behavior of the metamodel, each hyperparameter is perturbed in isolation. For demonstration purposes, the precise value of the perturbation is not important; instead, hyperparameters were perturbed to a round number a few orders of magnitude larger and smaller than the optimum value. The perturbation was chosen to illustrate the impact of the hyperparameter. By definition, these hyperparameter settings are *not* optimal, and the resulting metamodels are expected to be of little predictive value.

First, the θ_1 hyperparameter is increased to 1000.0, resulting in the hyperparameters listed in Table 11 and the metamodel depicted in Figure 53.

Table 11: Perturbed hyperparameters with increased θ_1

Hyperparameter	Value
θ_1	1000.0
θ_3	1.0019614×10^{-6}
r_x	0.20747739
r_a	0.08764052

The θ_1 hyperparameter controls the overall length scale of variation of the response. Dramatically increasing θ_1 has the effect of reducing the order of the metamodel. As contrasted with Figure 40, the somewhat cubic behavior in x is reduced to be nearly quadratic and the quadratic behavior in a is significantly flattened. Decreasing the θ_1 hyperparameter to 1.0×10^{-5} results in the hyperparameters listed in Table 12 and the metamodel depicted in Figure 54.

The response value shown in Figure 54 is visually unchanged from that corresponding to the optimum hyperparameters depicted in Figure 40; however, the variance displays unacceptable high-amplitude periodic behavior. This periodic behavior is believed to be caused by poor numerical behavior of the covariance matrix when θ_1 is very small. This phenomenon is most likely linked to the rounding error introduced by the finite precision

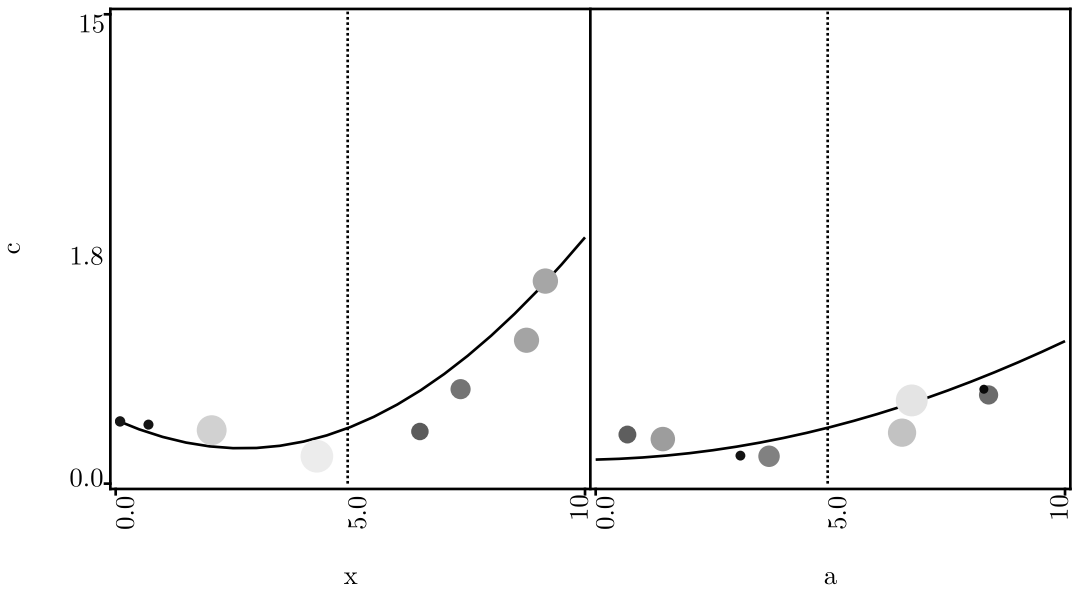

Figure 53: Metamodel resulting from increased θ_1

Table 12: Perturbed hyperparameters with decreased θ_1

Hyperparameter	Value
θ_1	1.0×10^{-5}
θ_3	1.0019614×10^{-6}
r_x	0.20747739
r_a	0.08764052

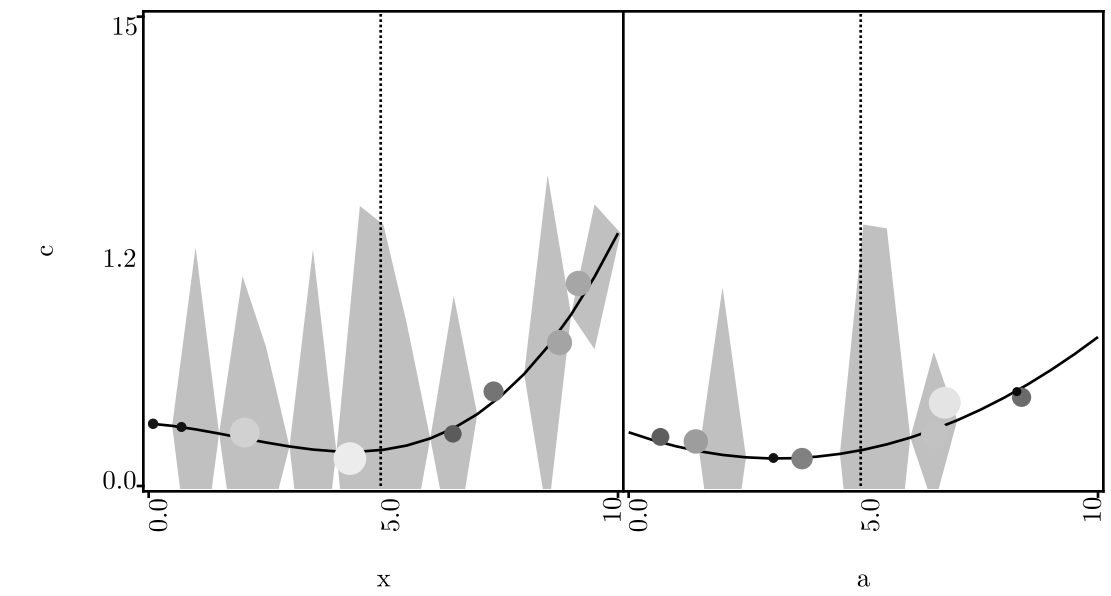

Figure 54: Metamodel resulting from decreased θ_1

storage of the training points in the data repository. Fortunately, if it is encountered during the optimization process, this phenomenon is easily recognized and alleviated by the user. The hyperparameter bounds constraint enforced by the optimizer could be increased to prevent this behavior from ever occuring. However, this behavior was seldom encountered and easily identified and rectified. Also, small values of the other hyperparameters were sometimes needed, and should not be disallowed. Finally, a crisp failure point which would have indicated an appropriate constraint setting was not observed.

Next, the θ_3 hyperparameter is increased to 0.1 resulting in the hyperparameters listed in Table 13 and the metamodel depicted in Figure 55.

Table 13: Perturbed hyperparameters with increased θ_3

Hyperparameter	Value
θ_1	0.004054227
θ_3	0.1
r_x	0.20747739
r_a	0.08764052

The θ_3 hyperparameter controls the scale of random noise expected in the data. A large

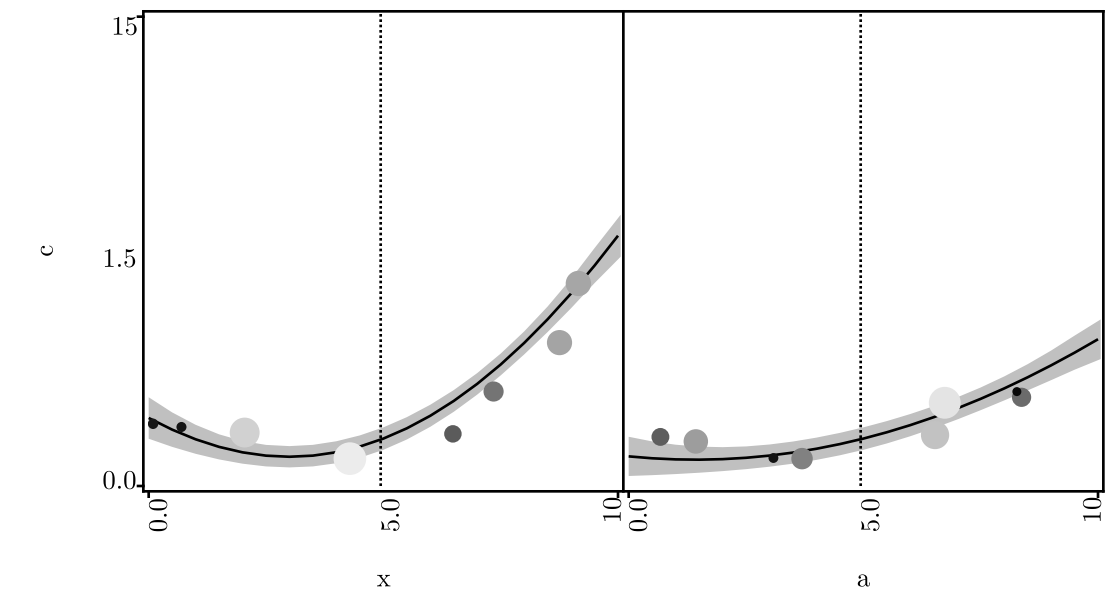

Figure 55: Metamodel resulting from increased θ_3

θ_3 will always lead to a large variance band, even when the data actually exhibits little noise. Notice that the order of the response has been reduced in a similar way as when θ_1 was increased in Figure 53. This result makes sense when the Occam's razor interpretation of Bayesian inference is considered; a more simple (lower order) model can explain the data in the presence of noise. The θ_3 hyperparameter was not reduced because its optimum value of 1.0019614×10^{-6} indicates that the optimizer's bounds constraint at 1×10^{-6} is already active. Reducing θ_3 below this level lead to numerical problems as discussed above. If a truly noise-free model was desired, it would be best to accomplish this by eliminating the noise model rather than by using a vanishingly small hyperparameter.

The r_a hyperparameter was then increased to 50, resulting in the hyperparameters listed in Table 14 and the metamodel depicted in Figure 56.

Table 14: Perturbed hyperparameters with increased r_a

Hyperparameter	Value
θ_1	0.004054227
θ_3	1.0019614×10^{-6}
r_x	0.20747739
r_a	50.0

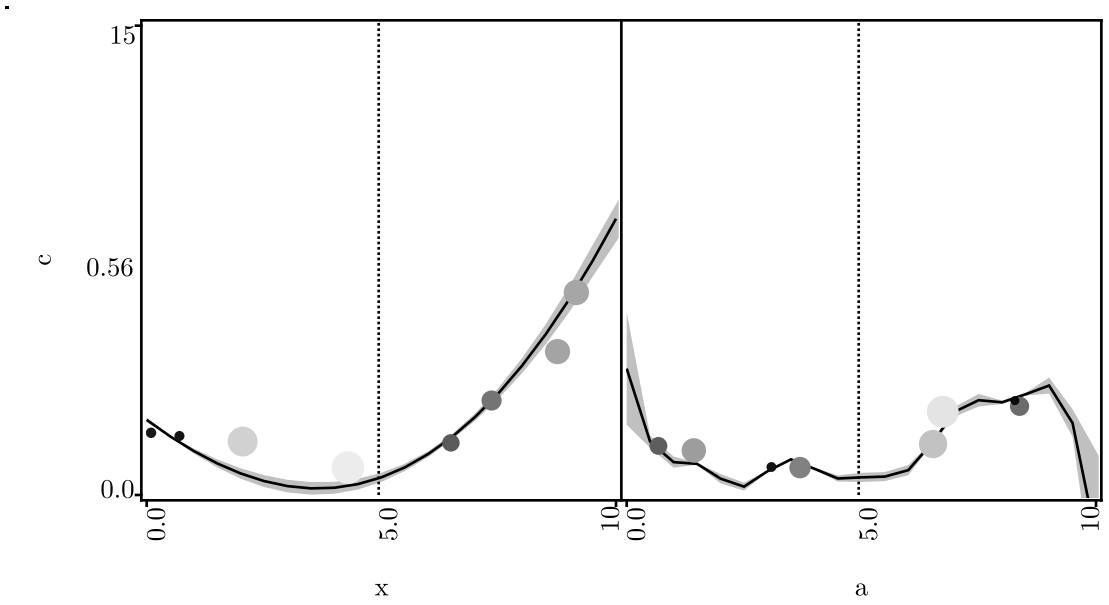

Figure 56: Metamodel resulting from increased r_a

The r_a hyperparameter controls the length scale of variation of the response with respect to input a. Recall that the hyperparameter definition was inverted from that typically used for numerical reasons. This hyperparameter may have been more aptly called a frequency rather than a length scale. Increasing the r_a hyperparameter greatly increases the frequency of variation of the response. The high order behavior in a allows the Gaussian process metamodel to account for the variation of the training data with lower order behavior in x. Although the depiction of the response in Figure 56 is somewhat jagged, this is an artifact of the sample resolution used for generating the plot; the underlying metamodel is still infinitely differentiable.

Decreasing the r_a hyperparameter to 1×10^{-4} results in the hyperparameters listed in Table 15 and the metamodel depicted in Figure 57.

As expected, decreasing the r_a hyperparameter decreases the order of behavior, resulting in a very flat response to changes in a. Similar experiments carried out on the r_x hyperparameter have very similar impact on the function response to changes in x and have not been included for brevity.

Table 15: Perturbed hyperparameters with decreased r_a

Hyperparameter	Value
θ_1	0.004054227
θ_3	1.0019614×10^{-6}
r_x	0.20747739
r_a	1.0×10^{-4}

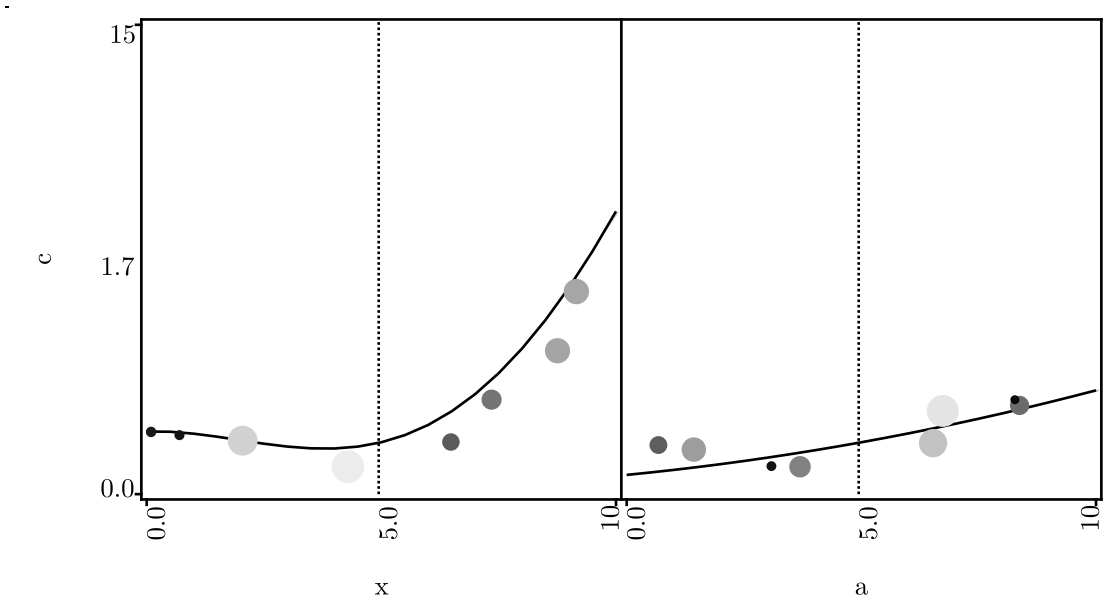

Figure 57: Metamodel resulting from decreased r_a

9.3.6 Training Data Variation

As mentioned in the introduction to this section, an excess of training data was generated for the purpose of these experiments. The remainder of this chapter serves to explore the impact of the size and composition of the training set on the Gaussian process metamodel. For these charts, the radius threshold has been set to 0.5 to ensure that all points are displayed.

The index limiting feature of the metamodel console was used to produce these variations. Therefore, each smaller dataset contains a subset of points from a larger dataset. For example, a ten point dataset contains all of the points from a seven point set (with three additional points). And a seven point dataset contains all of the points from a five point set. The initial thirty point Latin Hypercube sample has domain spanning properties; any subset is at best a random set of points.

Figure 58 depicts the metamodel corresponding to a ten point dataset with the baseline thirty point optimum hyperparameters. The resultant metamodel is visually indistinguishable from the thirty point metamodel of Figure 40; however, the response value reported at the hairlines has changed from 1.2 to 1.1. Due to rounding, the exact difference between the responses is not reported.

Figure 59 depicts the metamodel corresponding to a seven point dataset with the baseline hyperparameters. The x response appears to have diminished order and the variance bounds have increased near the bounds of the display. This can be indicative of insufficient information in that region, and if this behavior is observed in practice, the user should move the hairlines to that area and queue an additional sample point. The response at the hairlines is 1.2.

Figures 60 and 61 depict the metamodel corresponding to a six and five point dataset respectively (still with the baseline hyperparameters). At this point, the quality of the metamodel rapidly degrades. The elimination of the sixth training point (near $x = 10, a = 10$) causes the x response to change dramatically. The elimination of the sixth point leaves the $x > 5$ half of the domain completely unexplored.

The hyperparameters were next optimized for the seven point data set, resulting in the

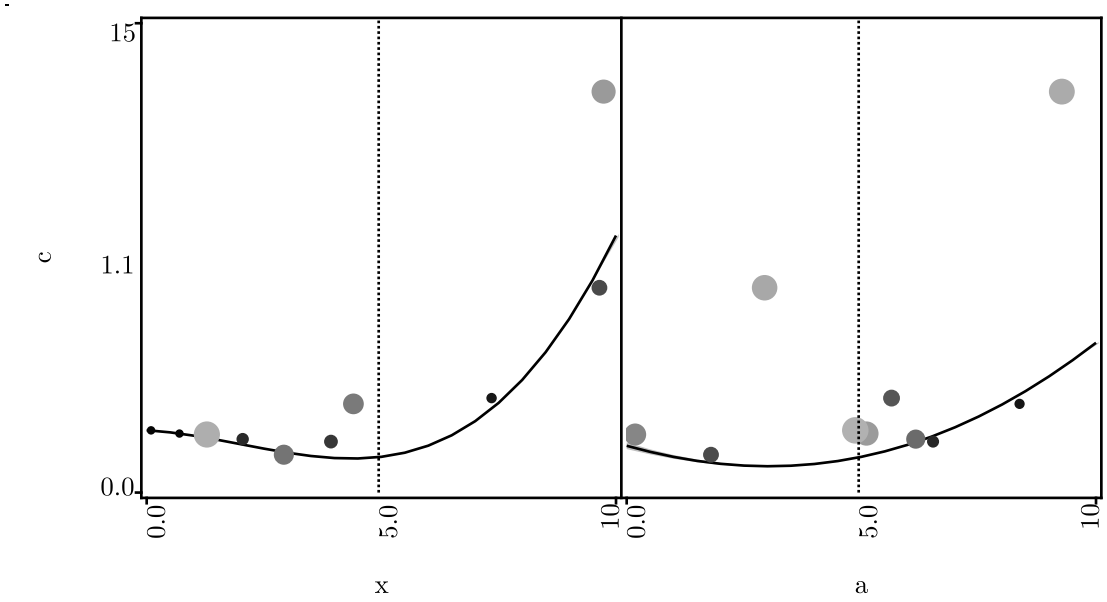

Figure 58: Metamodel based on ten training points

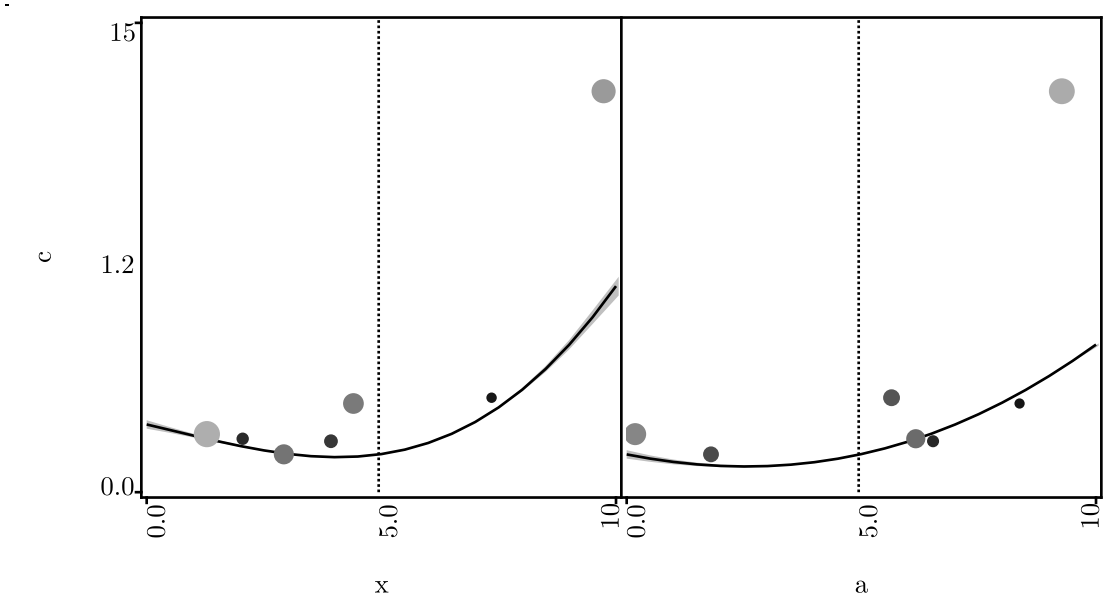

Figure 59: Metamodel based on seven training points

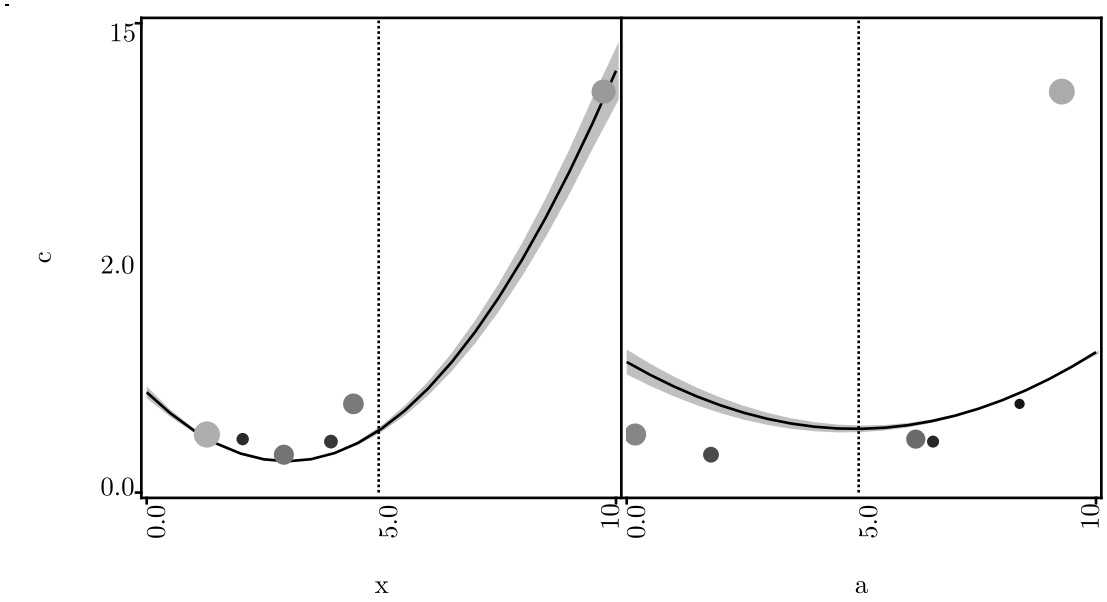

Figure 60: Metamodel based on six training points

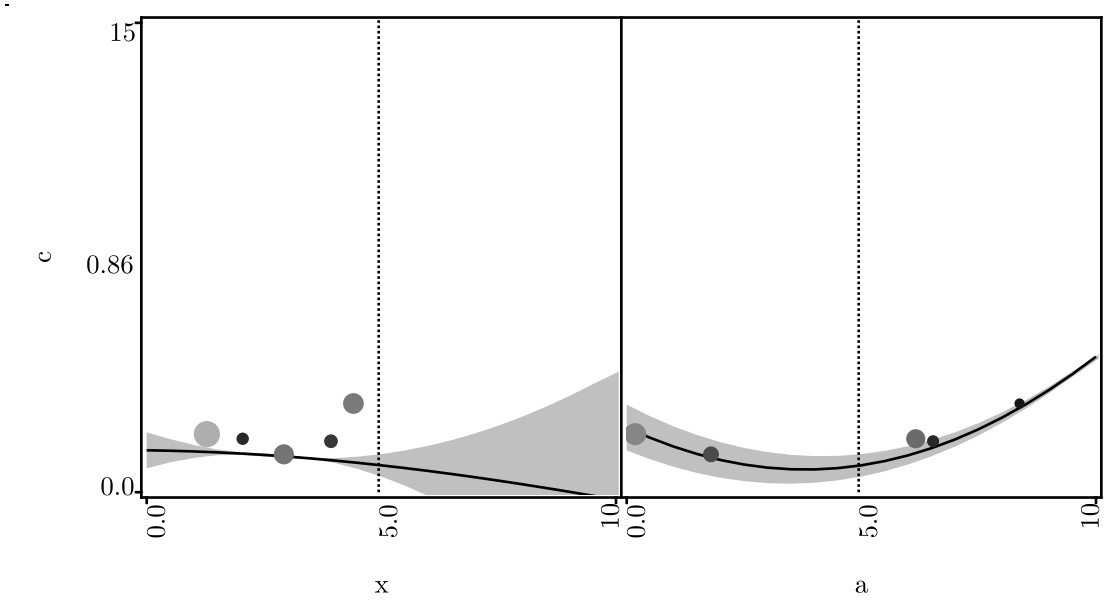

Figure 61: Metamodel based on five training points

hyperparameters listed in Table 16 and the metamodel depicted in Figure 62. While the x response strongly resembles the correct result, the a response bears little resemblance. The variance band is correspondingly larger.

Table 16: Optimized hyperparameters for seven training points

Hyperparameter	Value
θ_1	2.7575722
θ_3	6.3489075×10^{-5}
r_x	3.6317945
r_a	1.3664274

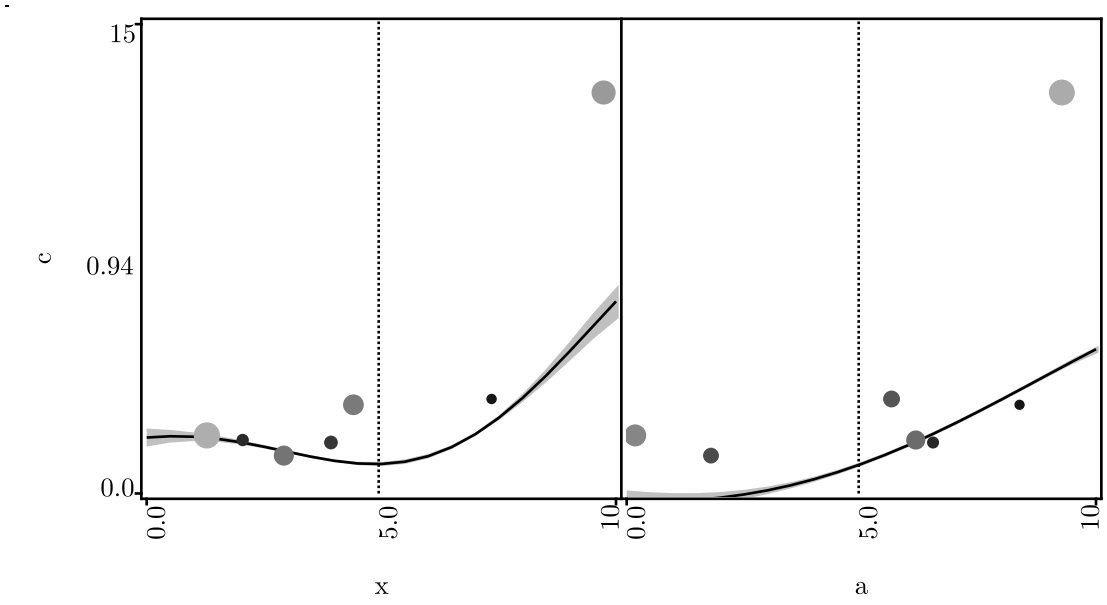

Figure 62: Metamodel based on seven training points with optimized hyperparameters

The hyperparameters were optimized for the five point data set, resulting in the hyperparameters listed in Table 17 and the metamodel depicted in Figure 63. In this case, the x response has nearly been eliminated. The extremely small value of the r_x hyperparameter (6.5384386×10^{-4}) corresponds to an extremely long length scale of variation, which infers constancy. Remarkably, in this case, the Gaussian process is able to completely explain the variation of the training data with variation in the a direction only.

It is likely that an improved distribution of points (without the need for *more* points)

Table 17: Optimized hyperparameters for five training points

Hyperparameter	Value
θ_1	22.879171
θ_3	1.9447591×10^{-4}
r_x	6.5384386×10^{-4}
r_a	3.0260715

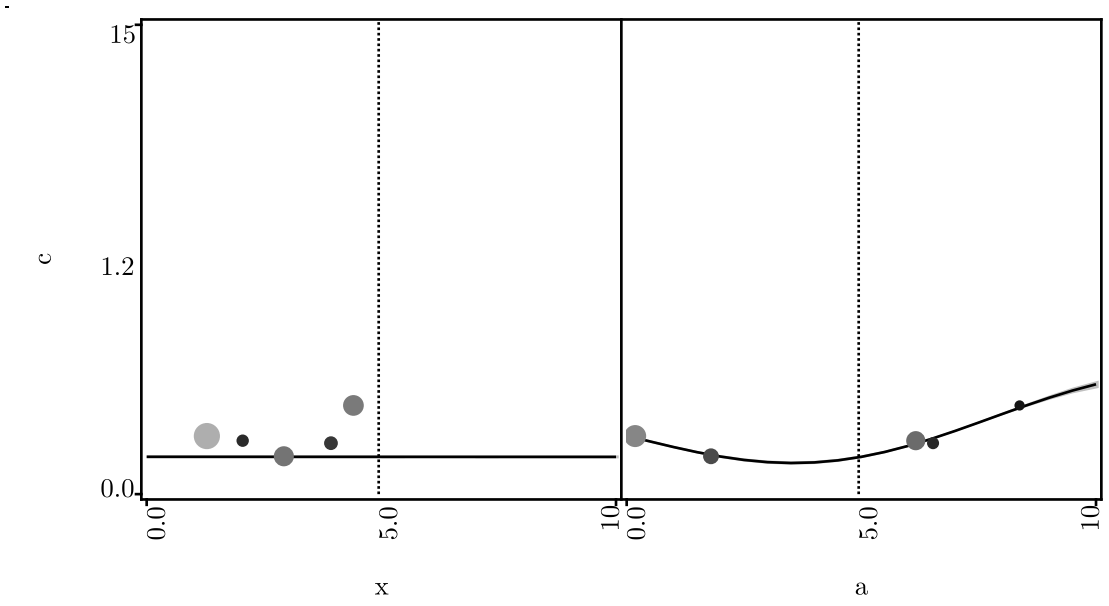

Figure 63: Metamodel based on five training points with optimized hyperparameters

would have resulted in improved metamodels for these last few cases. The consequences of the loss of the domain spanning property of the data set is best exemplified by Figures 60 and 61. If the initial thirty point set had been built from six five-point Latin Hypercube samples, then any multiple-of-five subset would have domain spanning properties. Any intermediate subset would formally be a random set of points, but have a strong resemblance to a domain spanning set. This approach may lead to better behavior when the required number of points is uncertain. Of course, extending this approach to its extreme (thirty one-point samples) destroys the domain spanning properties and results in a simple Monte Carlo sample.

CHAPTER X

DATA REPOSITORY

A database schema is presented in the spirit of the database architecture described in Chapter 6. The entire architecture was not implemented by the schema used in this research; instead, the implementation was limited to those portions required to accomplish the goals of the research. Even this subset of the entire architecture is capable of managing the data and metadata for most complex systems studies and represents a significant contribution.

The organization and structure of the data stored in a database is critical to the success of the database. What information may be stored and how it may be retrieved are largely dictated by the definition of the database. This definition is called the database schema. Databases do not store or access information in the same ways as a conventionally written application; this gives databases great power, but also makes them somewhat awkward to use. Database schema design comes with its own set of conventions and nomenclature appropriate to working with the features and pitfalls inherent. A brief understanding of these terms, given in the following paragraphs, is needed before a discussion of a database schema may begin. For a more in-depth discussion, please see the thorough theoretical treatment by Brathwaite [75] or the more pedagogical treatment by Harrington [76].

The relational data model is built of entities described as tables. An entity is used to store a set of related information, the attributes corresponding to the entity. The attributes that make up an entity form columns in the table. Each instance of an entity is a record that forms a row in the table. One of the consequences of database design is that all instances of an entity must have the same size. There can be no multivalued attributes. This means there is no direct way to include a list, array, heap, or any other container data structure with a non-fixed size.

Each record must have a unique identifier, called a primary key. While this primary key may have some meaningful value, it is generally better to make it an arbitrary, meaningless

number; this prevents consistency problems when a record changes. Other records may reference a particular record by including an attribute that will point to the record's primary key. This (the attribute in the referencing record) is called a foreign key. The ties between records represented by primary and foreign keys are called relations.

Relations provide a mechanism to look up one record based on the information in another record. A relation is a one-way mapping; there is no direct way to determine the records that reference a particular record. The referencing record may be indirectly determined by searching through all records to check if their foreign key matches the particular record. If a direct reverse lookup is required, the particular record may have a foreign key that references the record. This restricts the relation to a one-to-one relationship.

Many records may have foreign keys that reference a particular record. This provides for a one-to-many relationship. As before, the indirect lookup procedure requires a search to find all records which reference a particular record. It is through the multiplicity of relations that variable size data structures may be implemented in a database.

Often, it is necessary to provide for a many-to-many relationship between records. These are implemented by the addition of an intermediate entity, termed a pairing. A pairing provides for many-to-many relations between two records by providing a one-to-many relation to both records.

A complete diagram of the database schema developed for this research would be difficult to understand and too large for a single page. Instead, figures depicting related subsets of the schema have been included. For a reminder of the data architecture, see Figure 3 in Chapter 6. The schema subsets presented here parallel the architecture subsets presented in Chapter 6.

This schema may be used to describe a complex system of interacting tasks with accompanying variable fidelity metamodels. The analyses which implement the tasks and the servers which execute the analyses are also described by the schema, as are the various sources of error inherent to complex systems design.

In the schema diagrams, each box represents a table containing various attributes. Each table has a canonical and abbreviated name. Lines represent relations between tables. Lines

with an arrow represent the one-to-many relations by pointing from the primary key to the foreign key in the relation. The tables have been arranged to display the schema subsets in a compact form. Their spatial relationships have no meaning.

10.1 Task Interface

One of the defining steps to complex systems design is the decomposition of a complex system into subsystems and components. In the database schema, these are represented by tasks. Numeric values passed between components are represented by quantities. The definition of and relationship between tasks and quantities were highlighted in Figure 4 and are detailed in Figure 64.

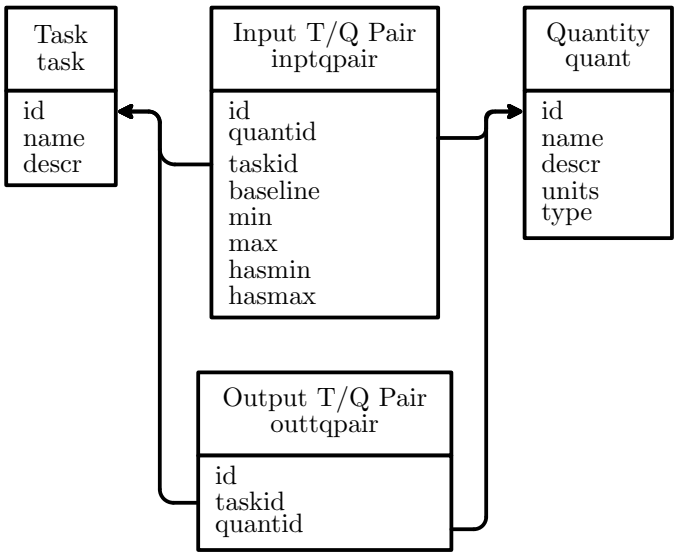

Figure 64: Task interface database schema

A task abstractly specifies an analysis requirement of the complex system, without specifying a particular code or theory. For example, estimating the aerodynamic performance of a wing, calculating the structural weight of a fuselage, or generating an engine deck are all tasks. As attributes, a task has an ID, a name, and a short description.

A quantity is a numeric value used as an input and/or an output to a task. As attributes, a quantity has an ID, a name, and a short description. Furthermore, the type (eg. integer or double) and units of the quantity are prescribed.

Tasks and quantities may be paired in two ways as depicted in Figure 64: as input and output task/quantity pairs. When a task is treated as a black box, the task/quantity pairs completely describe its interface. There can be many quantities associated with any task. A quantity may be associated with multiple tasks. The task/quantity pairs are used to establish these many-to-many relationships by creating a unique instance of an entity for each such pairing.

An input task quantity pair identifies a quantity as an input for a task. As attributes, a range of values with a baseline are prescribed. This range should be the largest physically sensible range for the variable. There can be many input variables for any task. A quantity may also be used as an input or output by another task.

An output task quantity pair identifies a quantity as an output for a task. There can be many outputs for a task. A quantity may also be used as an input or output by another task.

10.2 Implementations of a Task Interface

The task and its paired quantities combine to specify the interface for an abstract analysis requirement in a complex system. This interface can be seen as a contract which can be fulfilled by other entities. These task implementations can be used to fulfill the abstract analysis requirement in the complex system. A hierarchy of task implementations are represented by the database schema. These implementations were highlighted in Figure 5, and are detailed in Figure 65.

An analysis is a concrete implementation of a particular task. As an example, in the estimation of the aerodynamic performance of a wing, various analyses could be the lifting line theory, a vortex lattice method, a panel code, or a CFD code. As attributes, an analysis has an ID, a name, a short description, and an associated task. There can be many analyses for each task.

The metamodel and model/analysis pair entities have somewhat misleading names. The disconnect between their names and their function is a result of the historical development of this research, and while somewhat confusing, presents no problems or restrictions in

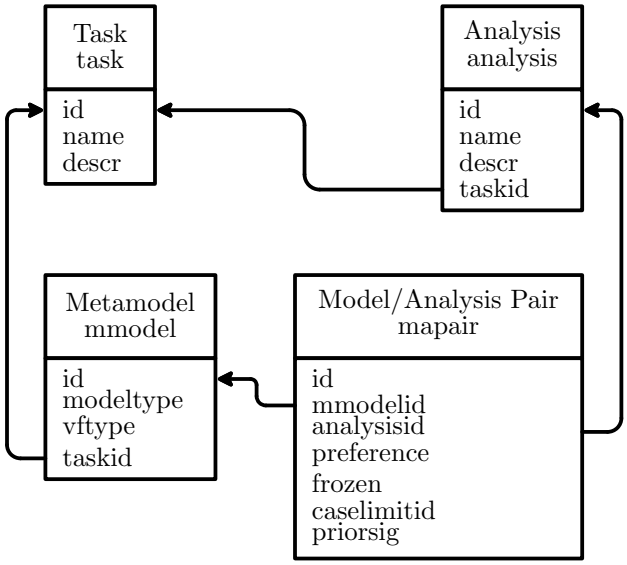

Figure 65: Task implementors database schema

generality. The metamodel entity represents a set of metamodels (in the conventional sense) which combine to form a variable fidelity metamodel. The model/analysis pair entity represents a metamodel in the conventional sense.

A metamodel is a variable fidelity approximation of a task. For example, if multiple analyses implementing a task are available, a variable fidelity metamodel dictates how the information is merged to produce a single estimate. As attributes, a metamodel has an ID and an associated task. Furthermore, the metamodel type (linear interpolation, Gaussian process, local least squares, etc.) and a variable fidelity scheme are prescribed. There may be multiple metamodels associated with any task.

A variable fidelity metamodel relies on multiple analyses and a single analysis may be used by multiple metamodels. The model/analysis pair is used to establish this many-to-many relationship. The model/analysis pair creates a unique entity for each such pairing.

This paring also includes some attributes beyond the simple pairing. In a variable fidelity situation, the preference parameter is used to identify a preliminary ranking of the fidelity of the analyses. The lowest ranked analysis is considered correct and other analyses are mapped to it. Aside from an absolute identification of the "master" code, the preference is simply a guideline, and may be overridden by the variable fidelity scheme.

Furthermore, some attributes are required to support the Gaussian process metamodel used in this research. These attributes are described fully below.

The variable fidelity capability included in this part of the schema is not intended to be used by this effort. It has been included as forward-looking infrastructure to facilitate future work. In a fixed fidelity metamodeling situation, only one model/analysis pair will exist, and the variable fidelity type attribute in the metamodel table will have no effect. Conceptually, a metamodel becomes an approximation for a particular analysis, and some redundancy for the identification of the associated task exists. The overhead introduced by these degenerate tables is negligible.

10.3 Analysis Support

The analysis table detailed above is supported by a computational resource and a record of every analysis execution as highlighted in Figure 6.

A server is a computing resource that will be used for analysis needs. As attributes, a server has an ID, a name, and a short description. Furthermore, the server's identity on the network (IP address and port), and access parameters (username and password) are prescribed.

One server may host multiple analyses. The same analysis may be hosted by multiple servers. The analysis/server pair is used to establish this many-to-many relationship by creating a unique entity for each such pairing. Analyses, servers, and analysis/server pair tables are depicted in Figure 66.

A case is a particular execution of an analysis. As attributes, a case has an id, the execution time, the runstate (-2, -1, 0, 1 for preparatory, queued, running, and completed respectively), and an associated analysis. There may be many executions of each analysis.

Each case was queued to be run at certain settings of the inputs which resulted, upon completion, in certain values of the outputs. These are stored in case/quantity pairs. Many cases may reference a particular quantity. A quantity may be referenced by many cases. The case/quantity pair is used to establish this many-to-many relationship by creating a unique entity for each such pairing and is depicted in Figure 66. The pairing also includes

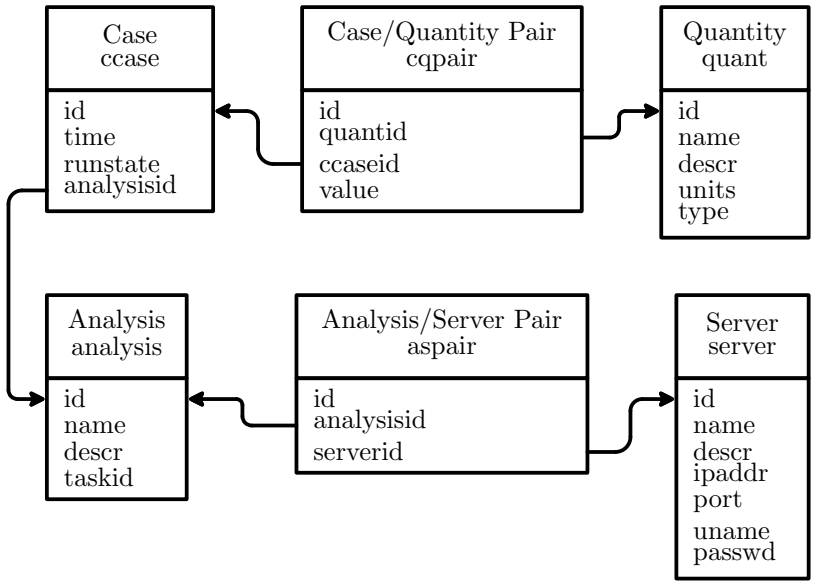

Figure 66: Analysis support database schema

the value of the particular quantity (input or output) for the particular case. For every completed case, there will be a complete set of quantities (input and output).

10.4 Support for the Metamodel

The metamodel table described above as a task interface implementation is an abstract entity that can represent any type of metamodel. In this research, Gaussian processes were used as the metamodel. Storing the information pertinent to a Gaussian process metamodel required direct consideration in the design of the database schema. These tables and relations were highlighted in Figure 7 and are detailed in Figure 67.

In addition to the attributes mentioned earlier, the model/analysis pair contains a flag describing the metamodel state as frozen or dynamic, the limiting id of the training data, and a parameter describing the width of the hyper-prior. The functionality of these parameters is discussed fully in the Gaussian process metamodeling chapter.

The Gaussian process requires further supporting information. Both the inputs and outputs have information stored in a pairing between the model/analysis and the appropriate quantity. The input pairing includes a minimum value and a range used for normalization. The response (output) pairing has a minimum and range for normalization as well as two

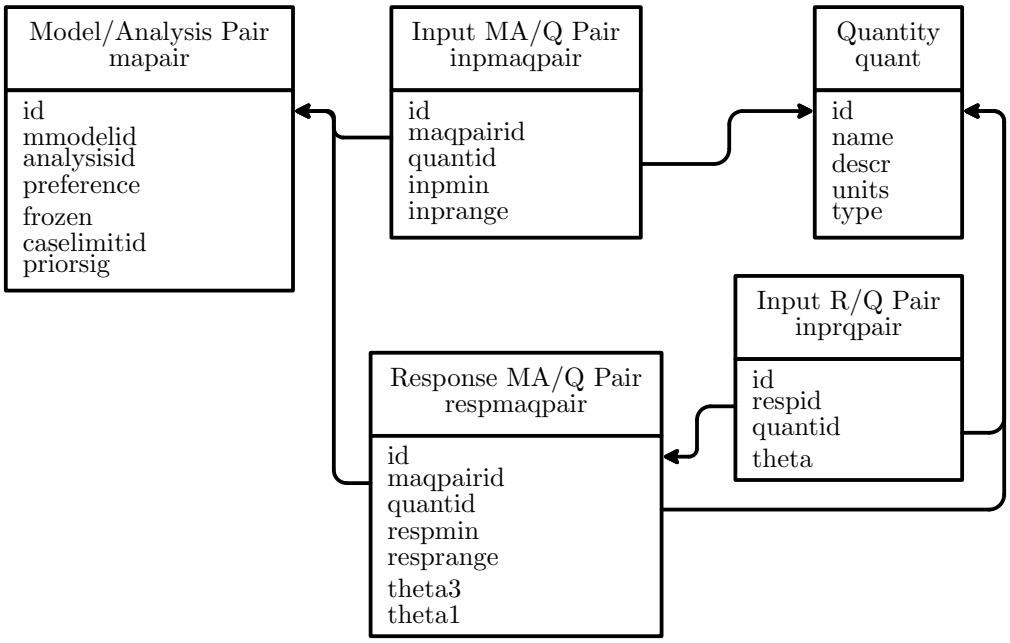

Figure 67: Metamodel support database schema

response-wide hyperparameters. The Gaussian process also requires a length scale hyperparameter associated with each input direction for every response. These are stored in a response/quanity pairing.

10.5 System Study

A study describes a complex systems design problem from a top level. As attributes, a study has an ID, a name, and a short description. Furthermore, the type (stochastic, domain spanning, or path building) of the study is prescribed. The study and its associated tables were highlighted in Figure 9 and are detailed in Figure 68.

In the same way that a complex systems design is made of component analyses, a study is made up of various tasks. Further, a task may be used by a multiple studies. The study/task pair is used to establish this many-to-many relationship. The study/task pair creates a unique entity for each such pairing.

There is a task/quantity pairing for all quantities involved with the black box definition of a task's interface. This, combined with the study/task pairings associated with a particular study, may be used to build a list of all quantities relevant to a study. This list

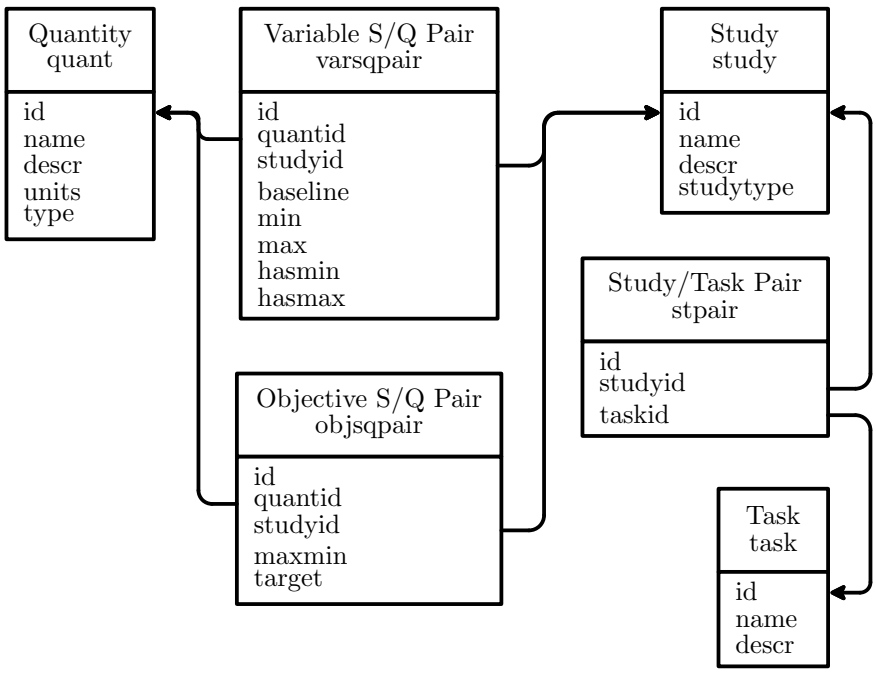

Figure 68: System study database schema

would include all variables that act as an interface to the study as well as all intermediate variables. Intermediate variables are inputs to a task which also appear as an output of another task. As such, they do not need an explicit value specified by the design process. Their value is provided by the output of another task.

A study/quantity pair is used to define the interface to a study, thereby system level inputs and notable outputs are identified. A study may have multiple quantities which define its interface. A quantity may be a part of the interface definition of multiple studies. The study/quantity pairs are used to define a unique entity for each such pairing.

A variable study/quantity pair identifies inputs at the complex system level. A study may be restricted to a subset of the variable bounds. Study variables identify variables to be manipulated by a particular study, as well as the range of interest and baseline for the purposes of the study. The range of interest should be a subset of the variables ranges associated with all the appropriate tasks.

An objective study/quantity pair identifies outputs of special interest at the complex system level. How they are to be treated by the design study exercise is identified by

whether it should be maximized, minimized, or driven to a target.

10.6 Error Source

Sources of error are attached to the quantity that they modify and to their source in the complex systems analysis. This source is either the study, or one of the task interface implementors. The error source and its associated tables were highlighted in Figure 10 and are detailed in Figure 69.

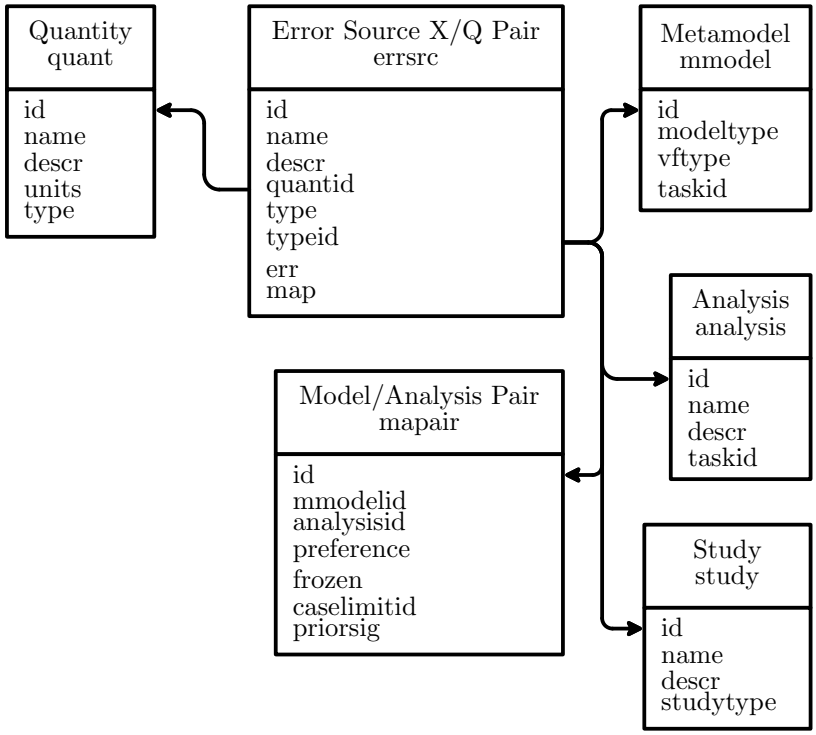

Figure 69: Error source database schema

Every error source has a name, description, and a foreign key identifying the associated quantity. The error table acts as a pairing between a quantity and an error source. This pairing is unique because the the error source can be one of four table types. To handle this variable-type relation, a type field indicates which table type the typeid foreign key refers to. Additionally, the error source stores a measure of the error magnitude and a flag indicating whether the error is mapped or uniform. Mapped error varies throughout the design space.

CHAPTER XI

SUPPORT COMPONENTS

The detailed workings of each of the support components of the new environment are discussed herein. These components (the experiment designer, executor, and setup GUI) have been designed to provide the minimum functionality required for this research. Significant improvements to the capabilities represented by these programs are readily available in the literature, and addressing the challenges these components address are active research areas.

11.1 Experiment Designer

A key element to any metamodeling process is choosing the sample points from which the model is built. It is the job of the experiment designer to interact with the other components and best lay out a strategy of choosing sample points.

The distribution of sample points for building a metamodel is frequently called a design of experiments (DOE). This name is carried over from the empirical (noisy) heritage of the field where the points to be considered were in fact experiments and selecting their quantity and distribution was in fact a design process. A DOE may refer to a strategy of point distribution or a specific set of selected points. The development of DOE strategies is a rich research field in itself and will only briefly be discussed here.

In most metamodeling processes, the sample points are selected a priori; no knowledge of the design space or a particular region of interest is known when the sample points are selected. In some new metamodeling processes, the points are dynamically adapted; sample points are added after some knowledge about the space and region of interest is known. This dramatically changes the best approach to distributing sample points. The framework developed herein is suitable for this kind of approach, but adaptation techniques were not the focus of this research.

11.1.1 Experiment Designs

Conventional DOE strategies are afforded no information about the function behavior or a specific region of interest within the design space. The sample point distribution must be determined based solely on knowledge of the behavior of the model and a statistical measure of the point distribution. In this way, a priori DOEs are a process of discovery, as they strive to provide maximum information on a response with minimal cost. There are a number of statistical techniques for measuring the quality of a point distribution which are briefly discussed below. Please see Box and Draper [31] for a classical discussion of DOE techniques and Simpson et al. [29, 77] for an up-to-date survey.

11.1.1.1 Point Distribution Quality

The ability of a DOE to provide information about the behavior of a space may be quantified by a variety of metrics. These metrics are based on a statistical analysis of the ability of a point set to provide information about a function's behavior in a space. They are not based on the behavior of the function in the space.

When discussing the ability of sample points to provide information for a fit, two types of error are considered. Variance and bias error combine to form the total error of an approximation to a function. Variance error accounts for the error encountered while carrying out the experimental investigation; it represents the deviation of the observations from the actual functional relationship. Bias error accounts for the fact that the chosen model is not the actual relationship; it represents the capability of the approximate model to match the actual functional relationship. In this research, there is no experimental error in the observations, therefore variance error is of limited utility.

Generally speaking, minimization of variance error leads to DOE points with maximum spread, which has the effect of smoothing experimental error. Conversely, minimization of bias error leads to DOE points having moderate spacing, far enough apart to represent the function over a significant range (rather than a point), yet close enough together to represent the function locally.

Unfortunately, most investigations into DOE quality focus on variance error at the

expense of bias error. While this simplifies the problem analytically, it is exactly the opposite of the appropriate simplification for this research. Bias error is difficult to consider because it requires some knowledge of the actual functional form being approximated. In studies that do consider bias error, an assumption that the function's true form is a polynomial of one higher degree than the approximating polynomial is frequently made.

Orthogonality is a metric that, when applied to DOEs, ensures the point distribution has minimum variance. Orthogonality makes no strong statement about the bias error of a DOE. Other measures of DOE quality including alphabetic optimality are typically limited in application to special cases. Due to their reduced generality, they must be applied with a great deal of care and will not be discussed in detail here.

11.1.1.2 A Priori Strategies

These quality metrics and various other factors have lead researchers to devise a variety of strategies for distributing sample points. These strategies vary in their applicability and complexity leading to their strengths and weaknesses.

An obvious point distribution strategy is that of a full factorial array. This amounts to a uniform grid to explore every combination of discrete settings of the variables. A full factorial approach yields uniform knowledge of the design space with excellent representation of boundary and corner regions. Because full factorial designs yield straightforward variable sweeps, the results may be intuitively interpreted, diagnostics performed, and trends identified before a model is fit. While a full factorial design is trivial to implement, the number of cases required grows exponentially with the number of variables; making this strategy very expensive for all but the smallest of systems. Various fractional factorial strategies exist to mitigate this cost, where a subset of the full factorial cases are selected.

More sophisticated approaches to point distribution are possible; the conception and evaluation of these strategies forms the core of the active DOE research field. Some of these strategies include block fractional, central composite, and Box-Behnken designs. While these strategies differ greatly, they share a common goal: to efficiently provide the information required to build the model. While sophisticated DOE strategies can reveal a great

deal of information about a model with very few points, they must be developed with a particular model in mind to ensure all the required effects are modeled. Because of DOE theory's empirical heritage, most DOE research focuses on small problems with relatively few variables. While some DOE theory can be extended to larger problems, creating large DOEs can be a computationally expensive task in itself.

Another very simple, but possibly not obvious, strategy also presents itself: sample points may be selected at random in a Monte Carlo approach. A Monte Carlo strategy will yield a uniform distribution of information and maintain orthogonality. A Monte Carlo approach has the unique characteristic that it may be stopped prematurely and still provide meaningful results throughout the design space; it may also be extended beyond the planned stopping point. A Monte Carlo strategy is trivial to implement and the implementation applies to a problem of any size.

A Monte Carlo approach to the starting problem yields a continuously uniform distribution of information throughout the design space. Furthermore, this approach can be terminated or resumed at will. As more cases are run, design space knowledge improves. Conversely, a traditional a priori DOE executed in a methodical manner results in an uneven distribution of information across the design space as the DOE is carried out and it must be finished to provide useful results.

Latin Hypercube sampling is an extension of Monte Carlo sampling designed to ensure domain spanning behavior without sacrificing the random nature of a Monte Carlo. A Latin Hypercube requires a priori knowledge of the number of sample points to be taken. A Latin Hypercube approach is simple to implement and has many desirable properties for sample point selection for these problems. An in depth discussion of Latin Hypercube and other sampling techniques is given in Koehler and Owen [33].

11.1.2 Implementation

The experiment designer component was designed to provide a simple yet powerful means of providing information for building metamodels. To this end, it supports four major modes of operation. These modes are classified by the way cases are queued (batch or individual),

and by the way the program is run (stand-alone or as an Analysis Server component). The user chooses whether the program is run stand-alone or as a component by the way the experiment designer is launched. Standalone operation is initiated just like any other program, by launching the executable. Component operation is initiated just like any other Analysis Server component, and details depend on the nature of the client. In the case of ModelCenter, the component is dragged from the server into the workspace.

Upon either mode of program launch, the program presents the user with a series of choices via a GUI as shown in Figure 70. First, the user chooses a database definition file through a familiar file open dialog. Once a database has been selected, the database is queried for a list of available analysis codes, and the list is presented to the user for selection. The user selects the program for which he wishes to design a set of experiments. The final startup choice made by the user is to choose whether he wishes to queue individual cases or batches of cases, thereby selecting the other component of the operational mode described above.

In stand-alone mode for queueing single cases, the user is presented with the interface depicted in Figure 71 labeled at the top with the active analysis code. The user is presented with text entry fields for each of the analysis program inputs. These fields are initialized with a reasonable default value based on the task interface of the analysis program. The user may edit these values and the queue the case by pressing the queue button. The user may queue as many cases as he wishes in this manner; when finished, the user terminates the program.

While completely functional, this mode of operation would quickly become tedious and is only expected to be used in special circumstances. The component mode for queueing single cases is depicted in Figure 72 with ModelCenter hosting the component. This mode can be used in much the same way as its stand-alone counterpart. However, it also provides the perfect mechanism for leveraging ModelCenter's built-in design of experiments techniques. Because the queuer is treated as any other Analysis Server component, any of ModelCenter's tools directly apply. In this context, this actually becomes a mechanism for queueing batches of cases.

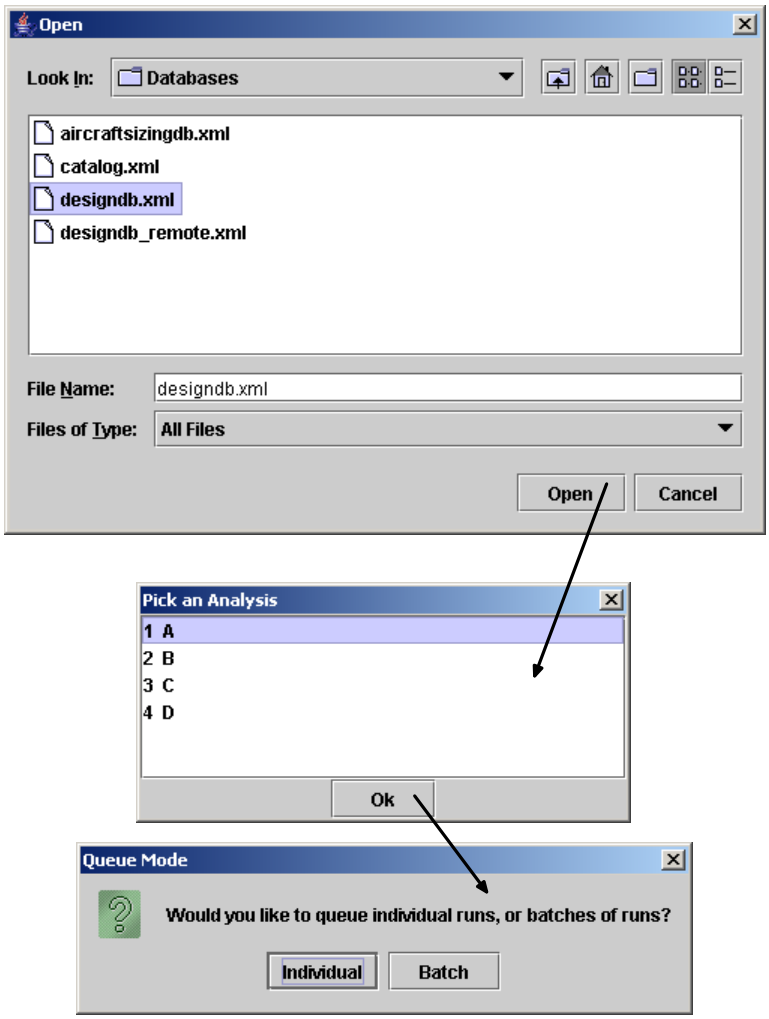

Figure 70: Experiment designer startup

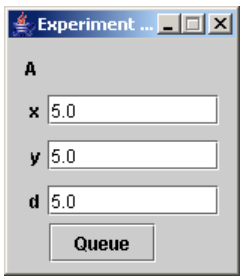

Figure 71: Experiment designer interactive single

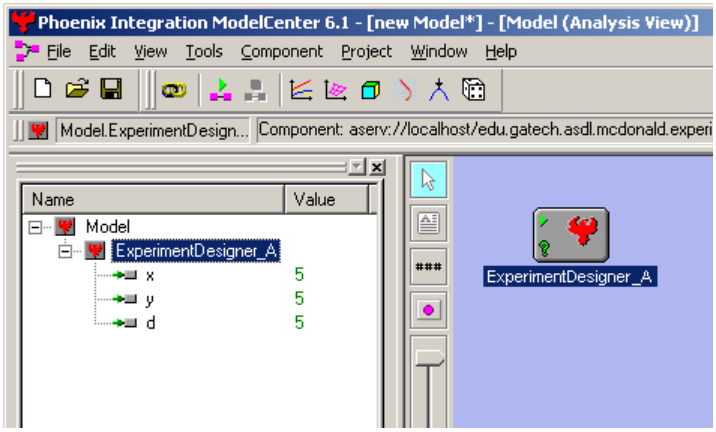

Figure 72: Experiment designer ModelCenter single

In stand-alone mode for queueing batches of cases, the user is presented with the interface depicted in Figure 73 labeled at the top with the active analysis code. At the center of the interface, there is a table of input variables with minimum and maximum values of each variable to be explored. These limits can be set purely manually, by the user, or guidance can be given by the limits imposed by a task interface, or by a study. This limit mode is selected through the pull down menu on the upper right. Obscured by, and directly below the pull down menu, there is a box to identify the study to be used to obtain the limits. There is also a button used to fetch a study from the database (action not shown). Usually, a design study does not specify limits on all of an analysis inputs. In such a case, the variables limited by the study are grayed out, while those left unaffected are left open to user entry. Once the user is satisfied with the selected limits, he enters the number of cases to be queued in the entry field in the lower left corner, and selects a point distribution mode (Monte Carlo or Latin Hypercube). Finally, the user clicks queue to prepare the batch of cases and queue them on the database. A small progress bar shows how much of the batch has been queued at any time.

The component mode for queueing batches of cases is depicted in Figure 74 with ModelCenter hosting the component. The same input functionality as the stand-alone batch mode is presented but in a ModelCenter interface, as was done with the single case mode. In most situations, the user will be better served with the stand-alone batch queuer; the

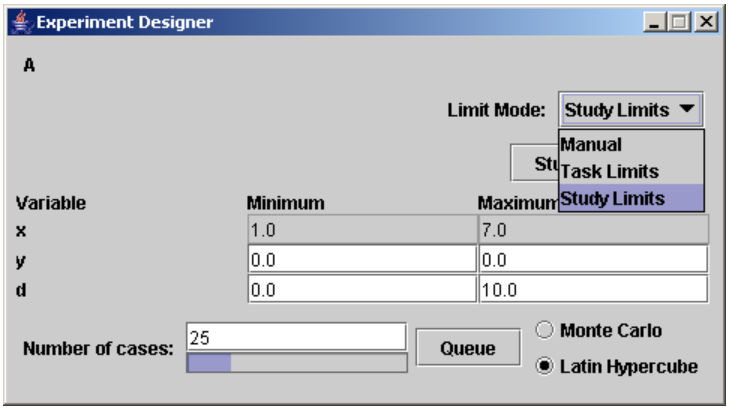

Figure 73: Experiment designer interactive batch

component version has been included for completeness.

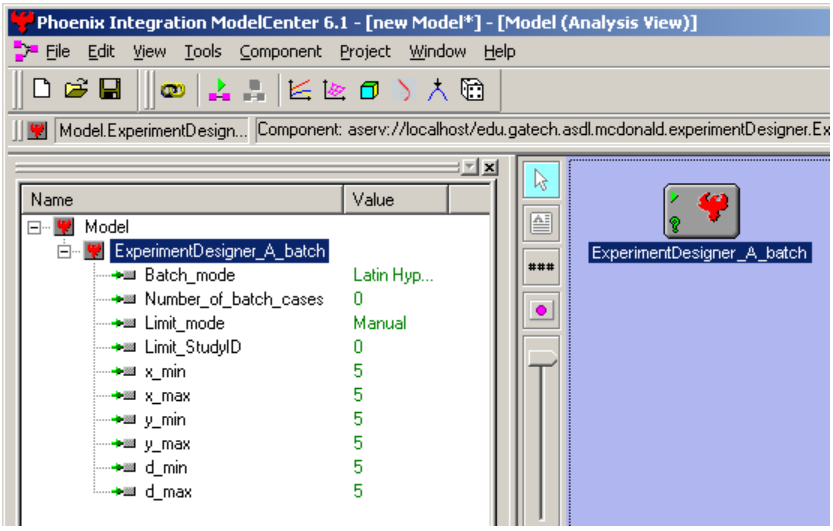

Figure 74: Experiment designer ModelCenter batch

11.2 Executor

Making sure that all queued cases are successfully executed while making efficient use of available computing resources is critical to the success of the complex systems design environment. This burden falls on the executor application. A simple technique to make sure computing resources are used efficiently is employed by this research.

Ongoing research in this area is popularly called grid computing or distributed computing while its roots trace back to the batch submission policies on the first multiuser computers. Efforts focus on optimal load balancing across diverse and distributed computing resources. Also, extensive efforts ensure the systems are fault tolerant to the dynamic computing and networking events that are inherent in such a system. Most of the tools that employ these techniques also use their own client/server architecture, and therefore are not compatible with Phoenix Integration's product line already selected as a foundation for this work. Recently, Phoenix Integration has released an advanced grid computing product, CenterLink Phoenix [78]. Unfortunately, this product was not available when this research started and therefore was not evaluated for use.

The primary focus of this work is on integrating error management and metamodeling into the complex systems design environment not on distributed computing. Therefore, the techniques employed by the executor are purposefully simple and rather naïve. Despite using the simplest approach possible, the executor is a multi-threaded application that is capable of monitoring the parallel execution of various analyses on dynamic, diverse, and distributed computing resources. A screenshot of the executor in action has been included as Figure 75.

In the upper left corner, under the heading *Queued Cases*, an execution queue is maintained. Cases are queued to be executed on a first in, first out (FIFO) basis. The queue is updated through occasional (and on-demand) polling performed by a dedicated thread. If the user requires an immediate update, clicking the *Refresh* button will trigger an event to update the queue.

In the lower left corner, under the heading *Available Resources*, a list of online computing resources is maintained. Under the subheading *Servers*, there is a list of all servers in the database that are online and reachable. Under the subheading *Analyses*, there is an aggregate list of analysis codes available on the online servers. These lists are maintained by occasional (and on-demand) polling performed by a dedicated thread. If the user requires an immediate update, clicking the *Refresh* button will trigger an event to update the lists.

On the right side, under the heading *Running Cases*, a list of cases currently executing

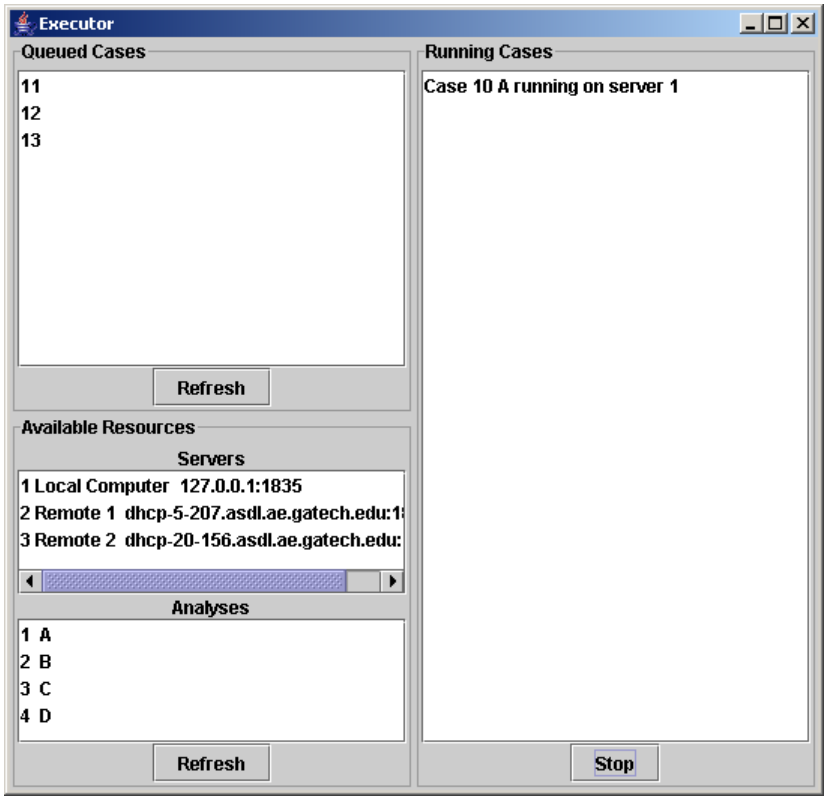

Figure 75: Executor GUI

is displayed. A dedicated thread monitors the online computing resources and the queued cases. When there is a match, a thread is spawned to manage the execution of a case. This check is performed occasionally and when triggered by other threads. If the user wants to stop new cases from being started, clicking the *Stop* button will prevent new cases from being started with no impact to cases already executing. When clicked, the *Stop* button will change to a *Start* button. Clicking the *Start* button will allow new cases to start.

Each executing case has a corresponding dedicated thread. The thread is responsible for handling every aspect of the execution of the case. This starts with retrieving the details of the case from the database and transferring those details to an analysis server. Then the thread marks the case as running in the database, starts the analysis executing, and waits for it to complete. Upon completion, the thread retrieves the results from the analysis server, transfers the results to the database, and marks the case as completed in the database. Finally, the thread triggers an event in the resource monitoring thread to

check for an available resource.

All the threads communicate as appropriate through events. When one thread needs access to a list maintained by another thread, an event is triggered to update the list, and then the list is accessed. In this way, the finite polling frequency will not normally cause any delay in code execution.

11.3 Setup GUI

Building a representation of a complex systems design problem in the structures provided by the database schema is a prerequisite for this approach to integrating metamodeling with the design environment. Manually entering all of the appropriate information to construct the required entities is potentially slow, tedious, and error prone. Even discovering some of the data is difficult. Once the entities exist, the relations connecting them must be constructed. These cross-references are based on matching values of arbitrary keys, to which no intuition applies. In order to efficiently and reliably set up and maintain the database representation of a complex systems design problem, there must be an interactive interface to elucidate the process. This burden falls on the setup graphical user interface (GUI) application.

The setup GUI provides an interface whose navigation should be intuitive to anyone familiar with computers today. Tabbed menus, dialog boxes, buttons, pick windows, etc. are used in the conventional manner. The proper use of these components however, requires a slightly more sophisticated user, one who is familiar with the database schema outlined in Chapter 10 and the client/server environment provided by Analysis Server.

While the trouble-free use of this application is approachable by anyone, this is not the foolproof interface expected from highly polished commercial software. There is neither online help nor error checking. There is no mechanism to encourage proper use of the interface, or to prevent the user from erring, and no way to recover when the user errs. These allowances have been knowingly made to simplify and speed development, and is an improvement over most prototype research software which typically has no interactive graphical interface. The setup GUI was developed because the dynamic nature of the problem necessitated it, not to imply a professional level of ease of use or developmental

maturity.

While the description and design of the database schema naturally flows from the big picture to the intricate details, the construction of the database flows from building blocks to a complex whole. The tabbed windows of the setup GUI roughly follow this paradigm to guide the user through the process. Each of the tabbed windows is described in Appendix D.

CHAPTER XII

DECOMPOSED SYSTEM CASE STUDY

In Chapter 1, a system of systems hierarchy was presented ranging from the national transportation system to the cooling flow circuit in a jet engine turbine. Systems design techniques are applicable to any system at any level of abstraction. This allows the designer to tailor the level of abstraction to the problem at hand. In demonstrating the utility and correctness of a systems design technique, it is not very important what specific system is considered. It is important that the example system exhibit challenges and traits representative of complex systems. For this research, a wide-bodied transport aircraft was chosen as the demonstration system. As a clear example of a complex system, the transport aircraft takes a pivotal role in the system of systems hierarchy described earlier. In the next section, the subsystems contributing to the aircraft system are discussed in detail. Then, in Section 12.2, the subsystems are assembled to model the aircraft system. Finally, in Section 12.3, a scenario conducted which demonstrates the effectiveness of the fidelity trade environment. This scenario acts as an experiment to test the guiding research hypotheses.

12.1 Tasks and Analyses

This complex system model is made of a series of tasks, each pertaining to a classical discipline of aerospace engineering. Each task defines an interface, a set of inputs and outputs. Any analysis which implements the task interface may be used in the system study. The task interfaces are documented in the following sections; the input and output quantities, their text name, units, range of validity, and a short description are given. The text name is given in addition to each quantity's symbol to facilitate some programs that do not support special characters and formatting.

Simple implementations of the task interfaces were developed as prototype analyses. The fidelity trade analysis can then be used to identify which task implementations must

be improved. The prototype analyses are documented in the following sections.

Gaussian process metamodels were fit to the prototype analysis implementations over a range of input variables appropriate for the example problem. A depiction of the response space for each analysis was included in the following sections. As with the metamodel and system explorer, this depiction represents each response as a row of charts with each input quantity providing a column. Each line may be interpreted as the variation of the response as one variable changes.

12.1.1 Aerodynamics

The aerodynamics task takes in information relating to the aircraft size, shape, and operating condition as listed in Table 18. The aerodynamics task produces an estimate of the aircraft drag polar at cruise and takeoff conditions as listed in Table 19.

Table 18: Aerodynamics task inputs

Quantity	Name	Min.	Max.	Units	Description
W	togw	0	–	lbf	Takeoff gross weight.
W/S	wos	0	–	lbf/ft^2	Wing loading.
l_{fuse}	lfuse	0	–	ft	Fuselage length.
$\mathcal{AR}$	ar	0	–	–	Aspect ratio.
$S_{wet,wing}$	swing	0	–	ft^2	Wing wetted area.
$S_{wet,fuse}$	sfuse	0	–	ft^2	Fuselage wetted area.
$S_{wet,add}$	sadd	0	–	ft^2	Additional wetted area.
σ	sigma	0	1	–	Density ratio at cruise.
θ	theta	0.75	1	–	Temperature ratio at cruise.

Table 19: Aerodynamics task outputs

Quantity	Name	Units	Description
$C_{D,0\ cr}$	cd0cr	–	Zero-lift cruise drag coefficient.
$C_{D,0\ sl}$	cd0sl	–	Zero-lift sea level drag coefficient.
e_{cr}	ecr	–	Oswald efficiency factor at cruise.
e_{sl}	esl	–	Oswald efficiency factor at sea level.

The methods of Roskam [79] were used to estimate the drag polar of the aircraft. This method involves semi-empirical equations as well as some coefficients estimated from charts.

The Oswald efficiency factor for the fuselage was estimated from Figure 2.5 on page 2.8 of the reference, and the coefficient value of 0.8 was chosen as representative of a round fuselage mated to a wing with an appropriate aspect ratio. For this coefficient, the value of 1.0 has no special meaning.

$$\Delta e_{fuse} = 0.8\frac{S_{fuse}}{S}$$

$$\Delta e_{other} = 0.05$$

The Oswald efficiency factor for the wing was calculated according to the method in Section 3.3.1 of Roskam [79], repeated in the equations below. Where R is the leading edge suction factor, taken to be 0.9 for this study.

$$e_{wing} = \frac{1.1\frac{C_{L_{\alpha_W}}}{\mathcal{AR}}}{R\frac{C_{L_{\alpha_W}}}{\mathcal{AR}} + (1-R)\,\pi}$$

$$C_{L_{\alpha_W}} = \frac{2\pi\mathcal{AR}}{2 + \sqrt{\frac{\mathcal{AR}^2\beta^2}{\kappa^2}\left(1 + \frac{\tan^2\Lambda_{c/2}}{\beta^2}\right) + 4}}$$

In these equations, β is the familiar compressibility term $\beta \equiv \sqrt{1-M^2}$ and κ is the ratio of the airfoil lift curve slope to the ideal two-dimensional value $\kappa \equiv C_{l_\alpha}/2\pi$. In this approach, the only difference between the cruise and sea level values for Oswald efficiency comes from the Mach number differences for each case.

The contributions to the Oswald efficiency factor are combined using the following equation given in Section 2.3 of Roskam [79].

$$e = \frac{1}{1/e_{wing} + \Delta e_{fuse} + \Delta e_{other}}$$

The skin friction drag of each component was calculated through the following standard empirical equation for turbulent skin friction coefficient taken from Schlichting [80] page 641. In this equation, Re_x is the component Reynolds number based on its reference length and the appropriate operating condition.

$$C_{D_f} = \frac{0.455}{(\log_{10} Re_x)^{2.58}}\frac{S_{wet}}{S}$$

The friction drag contribution of each component is calculated based on the appropriate reference length and wetted area. The zero lift drag coefficient was taken to be 25% more than the sum of the skin friction drag contributions as shown in the following equation. This factor accounts for various parasite drag sources which are otherwise difficult to estimate.

$$C_{D,0} = 1.25\left(C_{D_f,wing} + C_{D_f,fuse} + C_{D_f,add}\right)$$

Figure 76 depicts the Gaussian process metamodel of the aerodynamics analysis over a range of inputs appropriate for the example problem.

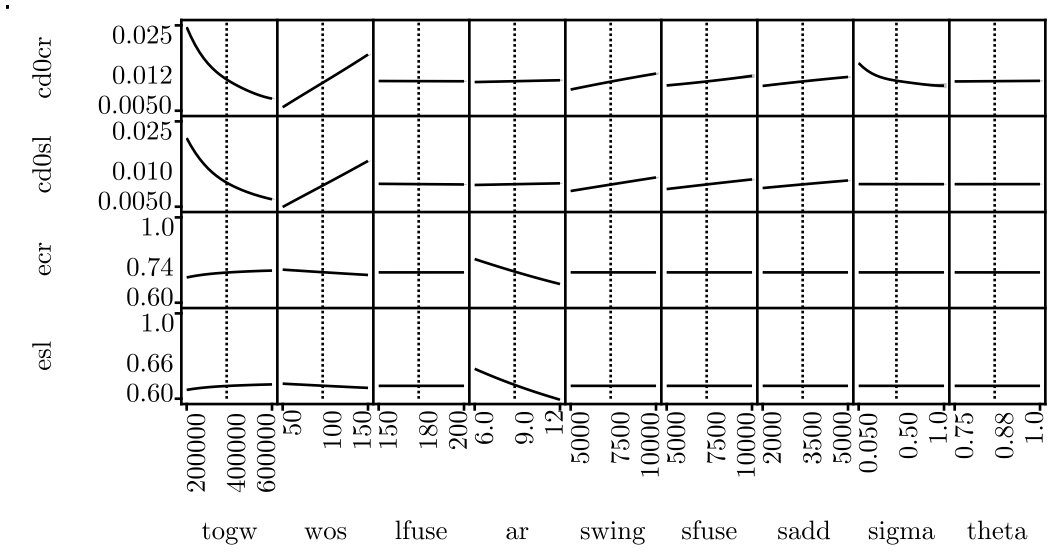

Figure 76: Aerodynamics analysis

This metamodel was created from a training data set of 60 points selected by a Latin Hypercube sample. In addition, 1000 test points were used to calculate the metamodel quality metrics described in Section 9.3. The resulting quality metrics for the aerodynamics metamodels were listed in Table 20.

The actual versus predicted response for the test points of each aerodynamics metamodel was plotted as Figure 77.

Figure 77 and Table 20 reveal that the $C_{D,0\ cr}$ metamodel is quite good, but clearly the worst of this set. The $C_{D,0\ cr}$ and $C_{D,0\ sl}$ responses should behave in a similar manner. However, the dependance of $C_{D,0\ cr}$ on the cruise altitude (defined by σ and θ) as seen in

Table 20: Metamodel Characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
$C_{D,0\ cr}$	5.52%	5.47%	0.988
$C_{D,0\ sl}$	1.33%	1.19%	0.9994
e_{cr}	0.027%	0.028%	0.99997
e_{sl}	0.033%	0.034%	0.99997

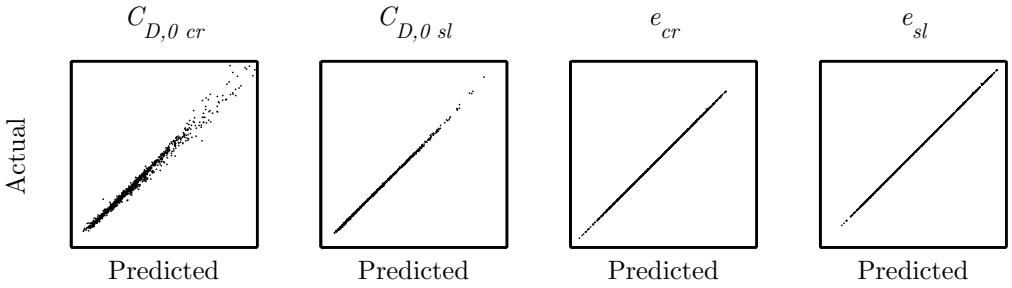

Figure 77: Aerodynamics metamodel

Figure 76 effectively increases the dimensionality of dependance for that response, which should require more training data for equal accuracy. This metamodel was judged sufficient, and additional training points were not added.

12.1.2 Atmosphere

The atmosphere task takes in the cruise density ratio as described in Table 21. It outputs the corresponding temperature ratio as described in Table 22. The density and temperature ratios are the ratio of the property at altitude to the property at sea level; because density and temperature decrease as you rise above sea level, both ratios will always be less than one.

Table 21: Atmosphere task input

Quantity	Name	Min.	Max.	Units	Description
σ	sigma	0	1	–	Density ratio.

A simplified routine based on the standard atmosphere was implemented [81]. First, the altitude corresponding to the supplied density ratio is calculated assuming a point in the

Table 22: Atmosphere task output

Quantity	Name	Units	Description
θ	theta	–	Temperature ratio.

troposphere using the following equation.

$$h = 145442\left(1 - \sigma^{1/4.255876}\right)$$

Then the corresponding temperature ratio is calculated while checking the validity of the troposphere assumption. If the calculated altitude is above the troposphere, the isothermal temperature ratio from the stratosphere is used. Consequently, this simple routine is not valid above the stratosphere; fortunately, the isothermal stratosphere layer extends to about 104,000 feet, well beyond the reasonable cruising altitude of this vehicle.

$$\theta = \begin{cases} 1 - \frac{h}{145442} & h < 36089 \quad \text{Troposphere} \\ 0.751865 & \quad \text{Stratosphere} \end{cases}$$

Figure 78 depicts the Gaussian process metamodel of the atmosphere analysis over a range of input appropriate for the example problem.

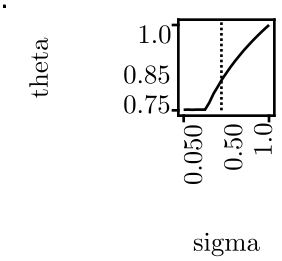

Figure 78: Atmosphere analysis

This metamodel was created from a training data set of 22 points. The initial training points were created with a Latin Hypercube sample. Additional points were manually added through the interactive metamodeling interface. In addition, 1000 test points were used to calculate the metamodel quality metrics described in Section 9.3. The resulting quality metrics for the atmosphere metamodel was listed in Table 23.

Table 23: Metamodel Characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
θ	0.35%	0.33%	0.99898

The actual versus predicted response for the test points of the atmosphere metamodel was plotted as Figure 79.

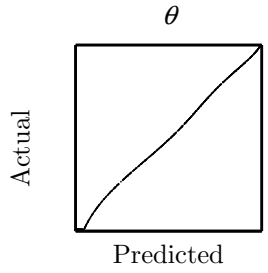

Figure 79: Aerodynamics metamodel

Figure 79 and Table 23 reveal that the θ metamodel is very good. It may seem excessive that 22 training points were required for a one-dimensional response. This was done to force the Gaussian process to model the cusp in the atmosphere model that occurs at the edge of the troposphere. The covariance function used in this thesis draws from the domain of infinitely differentiable functions; making it difficult to accurately model a cusp. A more efficient approach would be to implement a covariance function supporting spatially varying length scales or functions with discontinuous derivatives.

12.1.3 Geometry

The geometry task takes in information relating to the aircraft size and shape as listed in Table 24. The geometry task produces estimates of the wetted area of the major aircraft components as listed in Table 25.

The wetted area calculations used in FLOPS [82] were used. The equations repeated below are appropriate for subsonic transports. The wing root area not exposed due to the fuselage is accounted for. A factor is used to account for the airfoil shape and thickness. In these equations, c_r represents the root chord and b the wing span. There is also a factor to

Table 24: Geometry task inputs

Quantity	Name	Min.	Max.	Units	Description
W	togw	0	–	lbf	Takeoff gross weight.
W/S	wos	0	–	lbf/ft^2	Wing loading.
l_{fuse}	lfuse	0	–	ft	Fuselage length.
$\mathcal{AR}$	ar	0	–	–	Aspect ratio.

Table 25: Geometry task outputs

Quantity	Name	Units	Description
$S_{wet,wing}$	swing	ft^2	Wing wetted area.
$S_{wet,fuse}$	sfuse	ft^2	Fuselage wetted area.
$S_{wet,add}$	sadd	ft^2	Additional wetted area.

reduce the fuselage wetted area from the value of a similar circular cylinder.

$$S_{wet,wing} = (0.387\tau + 2)\left\{S - \frac{D_{fuse}\ c_r}{2}\left[2 - \frac{D_{fuse}}{b}(1-\lambda)\right]\right\}$$

$$S_{wet,fuse} = \pi\ D_{fuse}\,(l_{fuse} - 1.7D_{fuse})$$

$$S_{wet,add} = (0.387\tau + 2)\,(S_{vt} + S_{ht}) + S_{wet,nacelle}$$

An approximate value for the wetted area of the nacelles was assumed.

$$S_{wet,nacelle} = 800$$

Figure 80 depicts the Gaussian process metamodel of the geometry analysis over a range of inputs appropriate for the example problem.

This metamodel was created from a training data set of 12 points selected by a Latin Hypercube sample. In addition, 1000 test points were used to calculate the metamodel quality metrics described in Section 9.3. The resulting quality metrics for the geometry metamodels were listed in Table 26.

The actual versus predicted response for the test points of each geometry metamodel was plotted as Figure 81.

Figure 81 and Table 26 reveal that the geometry metamodels are excellent over most of the range of interest. The $S_{wet,wing}$ and $S_{wet,add}$ metamodels behave similarly and are not

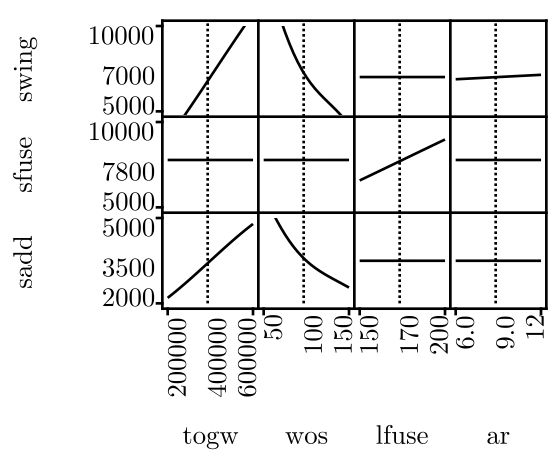

Figure 80: Geometry analysis

Table 26: Metamodel Characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
$S_{wet,wing}$	2.92%	4.48%	0.996
$S_{wet,fuse}$	0.014%	0.0139%	0.999998
$S_{wet,add}$	2.08%	3.68%	0.995

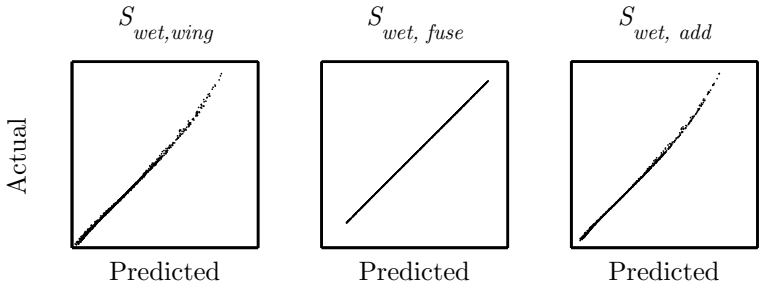

Figure 81: Aerodynamics metamodel

as good as the $S_{wet,fuse}$ metamodel. The calculation for $S_{wet,add}$ includes contributions due to the horizontal and vertical tail, which are dependant on the wing area. The performance of these metamodels degrades at large values of wetted area. A few additional points in the corresponding region of the design space as indicated by Figure 80 would most likely be sufficient to rectify this deficiency; however the accuracy of the metamodel was considered sufficient in the region of interest, and additional training data was not used.

12.1.4 Weights

The weights task takes in information describing the payload weight and mission required fuel fraction as listed in Table 27. The weights task produces an estimate of the aircraft takeoff gross weight as described in Table 28.

Table 27: Weight task inputs

Quantity	Name	Min.	Max.	Units	Description
W_p	wp	0	–	lbf	Payload weight.
W_f/W	wff	0	1	–	Fuel fraction.

Table 28: Weight task outputs

Quantity	Name	Units	Description
W	togw	lbf	Takeoff gross weight.

The following regression from Mattingly et al. [83] for the empty weight of a subsonic transport aircraft was used.

$$W_e/W = 1.02\ W^{-0.06}$$

The empty weight estimate was combined with the definition of the weights buildup given below. These two equations are iterated until convergence.

$$W = \frac{W_P}{1 - W_e/W - W_f/W}$$

Figure 82 depicts the weights analysis over a range of inputs appropriate for the example problem.

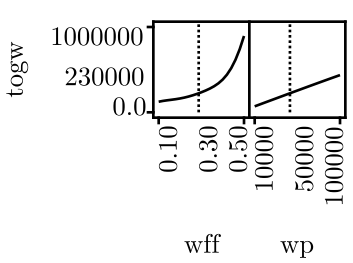

Figure 82: Weights analysis

This metamodel was created from a training data set of 15 points selected by a Latin Hypercube sample. In addition, 1000 test points were used to calculate the metamodel quality metrics described in Section 9.3. The resulting quality metrics for the weight metamodel were listed in Table 29.

Table 29: Metamodel Characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
W	4.04%	4.33%	0.997

The actual versus predicted response for the test points of the weight metamodel was plotted as Figure 83.

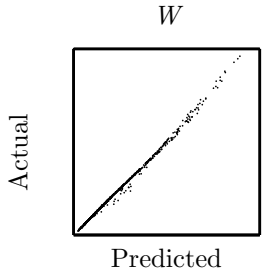

Figure 83: Aerodynamics metamodel

Figure 83 and Table 29 reveal that the weight metamodel is quite good over most of the range of interest. Closer examination of the residuals revealed that significant deviation was limited to $W_f/W > 0.45$. Additional data points in that portion of the design space would most likely reduce the error in that area; however, high fuel fractions are not considered

likely, and additional training data was not used.

12.1.5 Mission

The mission task takes in information relating to the aircraft size, cruise aerodynamics, propulsion, and mission parameters as listed in Table 30. The mission task produces an estimate of the required fuel fraction and cruise density as listed in Table 31.

Table 30: Mission task inputs

Quantity	Name	Min.	Max.	Units	Description
W	togw	0	–	lbf	Takeoff gross weight.
W/S	wos	0	–	lbf/ft^2	Wing loading.
$\mathcal{AR}$	ar	0	–	–	Aspect ratio.
θ	theta	0.75	1	–	Temperature ratio at cruise.
$C_{D,0\ cr}$	cd0cr	0	–	–	Zero-lift cruise drag coefficient.
e_{cr}	ecr	0	1	–	Oswald efficiency factor at cruise.
SFC	sfc	0	–	$lbm/hr/lbf$	Cruise specific fuel consumption.
R	range	0	–	nmi	Design range.
$W_{f,add}/W_f$	wffadd	0	1	–	Fuel fraction not used during cruise.

Table 31: Mission task outputs

Quantity	Name	Units	Description
W_f/W	wff	–	Fuel fraction.
σ	sigma	–	Density ratio at cruise.

Mission performance is modeled by calculating a weight fraction representing the fuel burned during the cruise leg of the mission. Cruise altitude is calculated such that the aircraft operates at best L/D at an average cruise weight. First, based on the following relation for a simple parabolic drag polar, we calculate the C_L for best L/D.

$$C_L{}^* = \sqrt{C_{D,0}\ \pi\ e\ \mathcal{AR}}$$

Then, the average cruise weight is calculated using the following equations. Fuel burned during takeoff and climb is assumed to be one third of the additional fuel load. Additional fuel is any fuel not used during cruise. This includes takeoff, climb, descent, loiter, landing,

and reserve.

$$W_{f,climb} = W_{f,add}/3$$

$$W_{ave} = W - W_{f,climb} - W_{f,cruise}/2$$

The cruising density and density ratio are calculated using the following simple relations based on the definition of the lift coefficient and the density ratio.

$$\rho = 2W_{ave}/\left(S\ C_L^*\ V^2\right)$$

$$\sigma = \rho/\rho_{sl}$$

The weight fraction consumed during cruise is calculated using the following form of the Breguet range equation. Total fuel weight is calculated as the sum of cruise fuel and additional fuel.

$$\frac{W - W_{f,climb} - W_{f,cruise}}{W - W_{f,climb}} = \exp\left(\frac{-SFC\ D\ t}{W_{ave}}\right)$$

$$W_f = W_{f,cruise} + W_{f,add}$$

The additional fuel load is specified as a fraction of the total fuel load. Consequently, the mission performance must be iterated until convergence to ensure consistancy.

Figure 84 depicts the mission analysis over a range of inputs appropriate for the example problem.

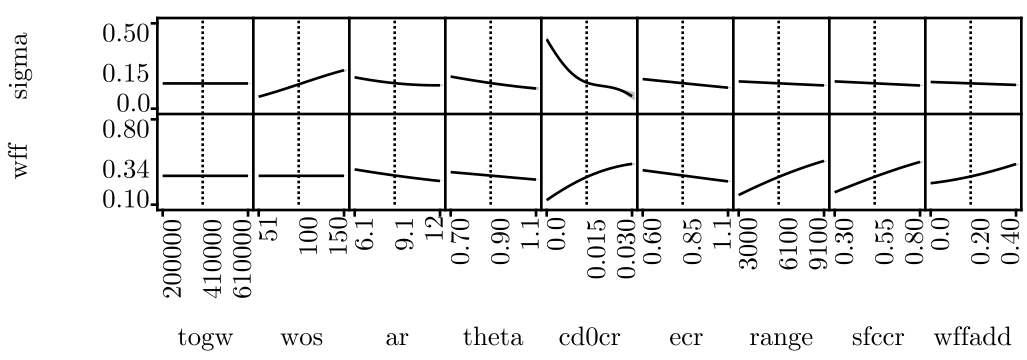

Figure 84: Mission analysis

This metamodel was created from a training data set of 45 points selected by a Latin Hypercube sample. In addition, 1000 test points were used to calculate the metamodel quality metrics described in Section 9.3. The resulting quality metrics for the mission metamodels were listed in Table 32.

Table 32: Metamodel Characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
σ	4.55%	4.19%	0.990
W_f/W	2.27%	2.07%	0.997

The actual versus predicted response for the test points of the mission metamodels were plotted as Figure 85.

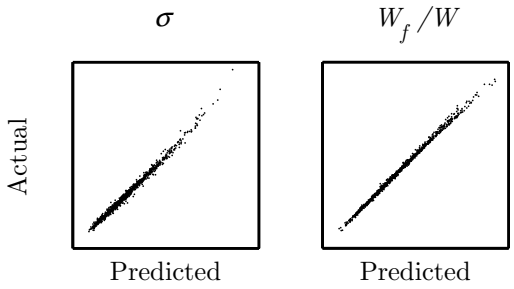

Figure 85: Aerodynamics metamodel

Figure 85 and Table 32 reveal that the mission metamodels are very good over the range of interest.

12.1.6 Point Performance

The performance task takes in information relating to the aircraft size, aerodynamics, and operating condition as listed in Table 33. The performance task produces an estimate of the point performance of the aircraft as listed in Table 34.

Very simple takeoff and landing distance calculations are performed according to the method outlined in Anderson [84]. In the derivation of this equation, forces are assumed constant throughout takeoff for the purposes of integration. Representative values corresponding to those that occur at 70% of takeoff speed were used. A rolling friction coefficient

Table 33: Performance task inputs

Quantity	Name	Min.	Max.	Units	Description
W	togw	0	–	lbf	Takeoff gross weight.
W/S	wos	0	–	lbf/ft^2	Wing loading.
T/W	tow	0	–	–	Thrust to weight ratio.
$\mathcal{AR}$	ar	0	–	–	Aspect ratio.
$C_{L,max}$	clmax	0	–	–	Maximum takeoff lift coefficient.
σ	sigma	0	1	–	Density ratio at cruise.
θ	theta	0.75	1	–	Temperature ratio at cruise.
$C_{D,0\ cr}$	cd0cr	0	–	–	Zero-lift drag coefficient at cruise.
e_{cr}	ecr	0	1	–	Oswald efficiency factor at cruise.
$C_{D,0\ sl}$	cd0sl	0	–	–	Zero-lift drag coefficient at sea level.
e_{sl}	esl	0	1	–	Oswald efficiency factor at sea level.
W_f/W	wff	0	1	–	Fuel fraction.

Table 34: Performance task outputs

Quantity	Name	Units	Description
S_{TO}	tofl	ft	Takeoff field length.
S_{LDG}	ldfl	ft	Landing field length.
$P_{s,cr}$	pscr	ft/min	Excess power at cruise.
$P_{s,sl}$	pssl	ft/min	Excess power at loiter.

of $\mu = 0.02$ was assumed for dry pavement. A standard assumption of liftoff velocity 20% higher than the stall speed was made. The equation for takeoff distance is given below. If the thrust margin at takeoff was found to be less than 10%, the case was rejected because the aircraft could not take off in a reasonable distance.

$$S_{TO} = \frac{1.44\ W^2}{g\ \rho\ S\ C_{L,max} \left\{ T - \left[D + \mu \left(W - L \right) \right]_{0.7V_{LO}} \right\}}$$

Approach speed was taken to be 30% higher than the stall speed for landing. Landing integration is once again calculated based on constant forces taken at a representative value corresponding to the forces occurring at 70% of approach speed. A braking friction coefficient of $\mu = 0.4$ was assumed for dry pavement. The $C_{L,max}$ in the landing configuration is taken to be 10% higher than the takeoff value. The equation for landing distance is given below.

$$S_{LDG} = \frac{1.69\ W^2}{g\ \rho\ S\ C_{L,max} [D + \mu \left(W - L \right)]_{0.7V_T}}$$

The definition of specific excess power, given below, was used to calculate point performance at cruise and loiter conditions. The loiter condition is calculated at Mach 0.5 at sea level conditions at takeoff weight. The cruise condition was taken to be that for maximum L/D at cruise Mach with half of the fuel supply burned.

$$P_s = \frac{T - D}{W} V$$

The thrust lapse for the engine was approximated with the following relation for a high bypass turbofan given by Mattingly et al. [83].

$$T/T_{sl} = \left[0.568 + 0.25 \left(1.2 - M \right)^3 \right] \sigma^{0.6}$$

Figure 86 depicts the performance analysis over a range of inputs appropriate for the example problem.

This metamodel was created from a training data set of 75 points selected by a Latin Hypercube sample. Cases with a very low thrust margin were omitted, resulting in a training data set of 53 points. In addition, 1000 test points were generated to calculate the

Figure 86: Performance analysis

metamodel quality metrics described in Section 9.3. These test points were also filtered, resulting in 693 test points. The resulting quality metrics for the performance metamodels were listed in Table 35.

Table 35: Metamodel Characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
S_{TO}	6.92%	6.43%	0.984
S_{LDG}	2.06%	2.42%	0.9987
$P_{s,cr}$	6.06%	0.67%	0.9998
$P_{s,sl}$	16.74%	2.14%	0.9991

The actual versus predicted response for the test points of each performance metamodel was plotted as Figure 87.

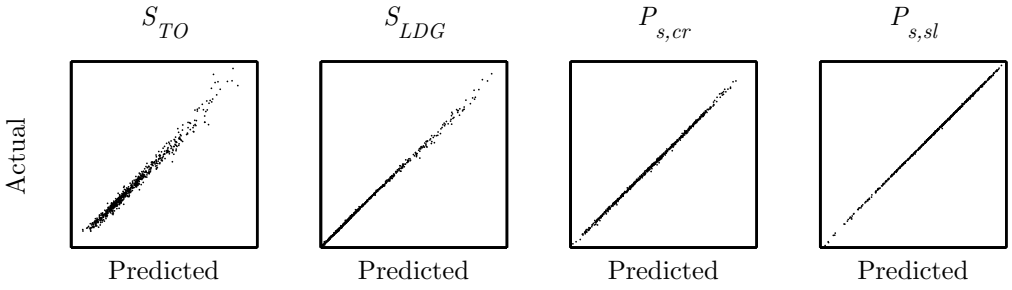

Figure 87: Aerodynamics metamodel

Figure 87 and Table 35 reveal that the performance metamodel is quite good over most of the range of interest for each of the responses. From Figure 87, the S_{TO} metamodel appears to be the worst, but its RMS error is less than 7%. The error metrics in Table 35 highlight the need for two RMS measures of error. In situations where quantities in the sample set can be near zero, the relative RMS error, RMSE_{x_i}, can mislead as to the quality of the metamodel. In this situation, the RMS error normalized by the mean, $\text{RMSE}_{\bar{x}}$, is a more appropriate quality measure.

12.2 Aircraft System

The complex system model representing the aircraft is assembled from the tasks described in the previous section. The method described in Section 8.1.1 was used to determine the

order of the tasks. Any quantities not specified as outputs from a task are treated as system level inputs. The resulting system is shown in Figure 88.

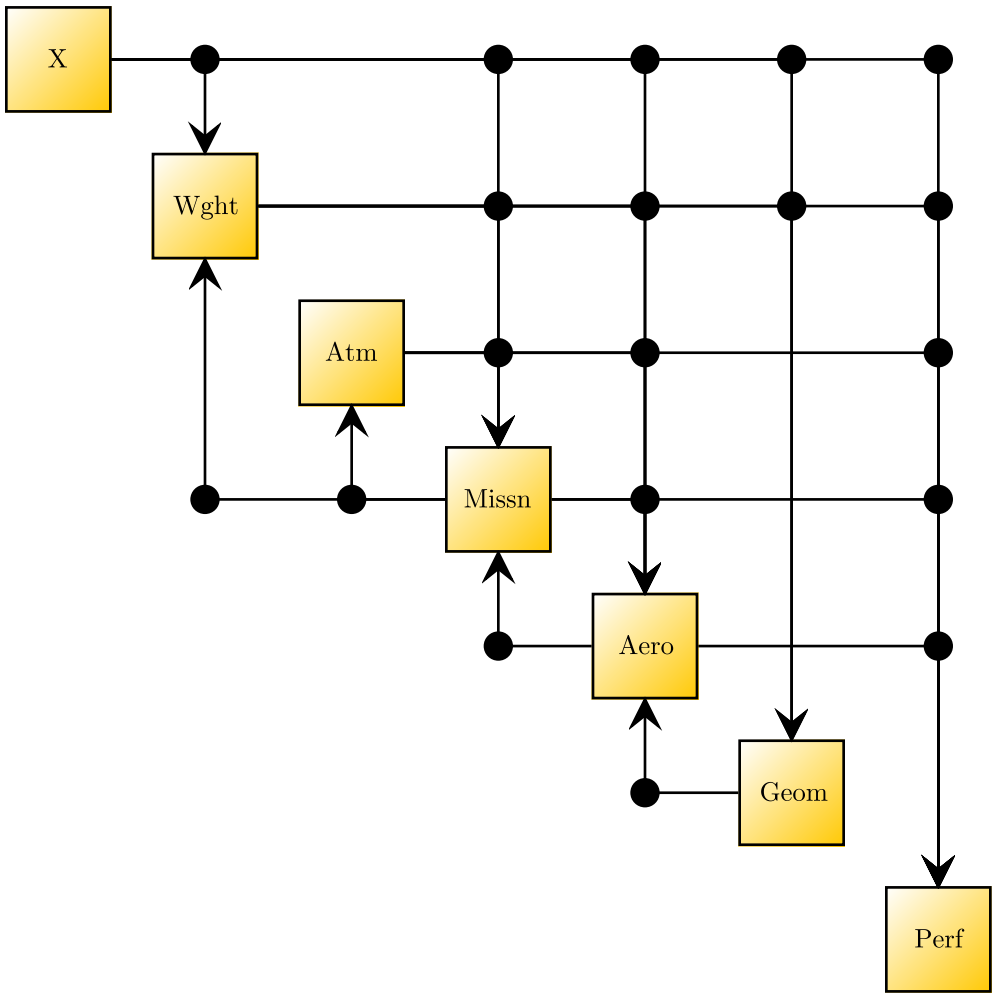

Figure 88: Example system DSM

Ranges of interest and baseline values for the system level inputs were selected to represent a large design space of alternatives as listed in Table 36.

Additionally, some quantities that would qualify as system level variables were set to representative values and held constant throughout the study. This effectively hides these quantities from the systems perspective. This was done to limit the scope of the study and the resulting visualizations to a manageable level. These implied variables and their settings are listed in Table 37.

Table 36: System level variable ranges and settings

Quantity	Min.	Max.	Baseline	Units
W_p	10,000	100,000	55,000	lbf
W/S	50	150	100	lbf/ft^2
l_{fuse}	150	200	175	ft
$\AA$	6	12	9	–
SFC	0.3	0.7	0.5	$lbm/hr/lbf$
R	3000	9000	6000	nmi
$W_{f,add}/W_f$	5	30	17.5	%
T/W	0.18	0.3	0.24	–
$C_{L,max}$	1.0	4.0	2.5	–

Table 37: Implied variable settings

Quantity	Value	Units
M_{cr}	0.83	–
$\Lambda_{c/2}$	25°	
D_{fuse}	17	ft
S_{VT}/S	10	%
S_{HT}/S	23	%
λ	0.25	–
τ	12	%

The baseline settings of the system level variables result in a converged vehicle represented by the system outputs listed in Table 38. An isometric view of a three-dimensional model of the baseline aircraft is given in Figure 89; top, side, and front views are given in Figures 90–92.

Table 38: Sized vehicle characteristics

Quantity	Value	Units
W	310000	lbf
σ	0.193	–
W_f/W	34.5	%
θ	0.752	–
$S_{wet,wing}$	5325	ft^2
$S_{wet,fuse}$	7800	ft^2
$S_{wet,add}$	2880	ft^2
$C_{D,0\ cr}$	0.0151	–
$C_{D,0\ sl}$	0.0113	–
e_{cr}	0.735	–
e_{sl}	0.655	–
S_{TO}	5430	ft
S_{LDG}	4200	ft
$P_{s,sl}$	3360	ft/min
$P_{s,cr}$	390	ft/min

A system explorer view of the entire aircraft design space is included as Figure 93. The hairlines represent the baseline values of the system level variables. The curves represent the variation of the system response to the change of a single system level variable. Each point contributing to a curve in the system explorer represents a converged vehicle model.

Some representative constraints were placed on the point performance of the aircraft as listed in Table 39. These constraints were plotted on a customary T/W vs. W/S view of the design space as shown in Figure 94. The vertical and horizontal hairlines represent the baseline vehicle. As expected, the baseline vehicle satisfies the constraints.

12.3 Error Management

In order to demonstrate the utility of the fidelity trade environment, a representative error management scenario is carried out on the example transport aircraft system. The focus of this example is to illustrate a possible fidelity decision making process, so the point of

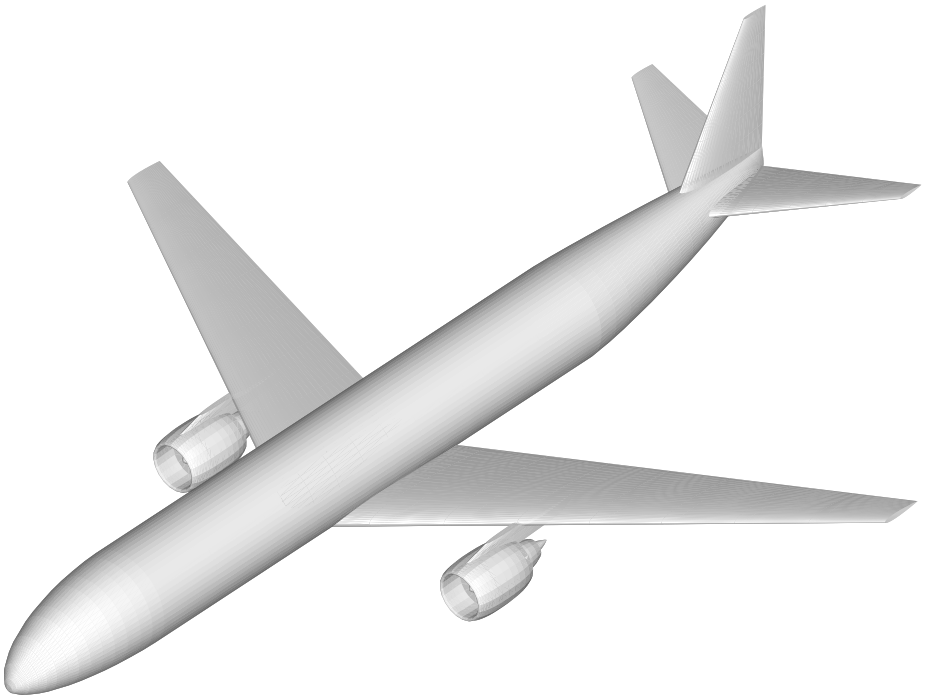

Figure 89: Isometric view of baseline aircraft

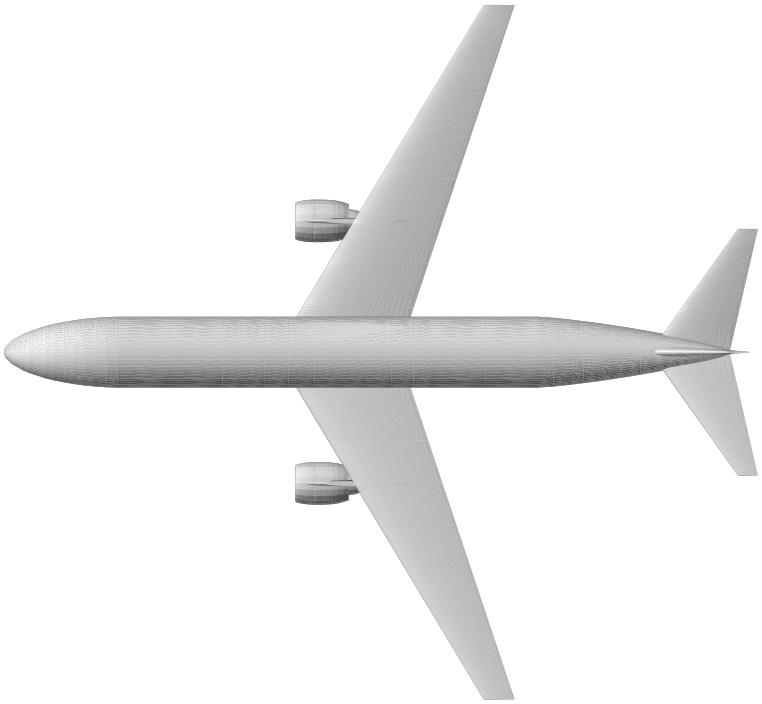

Figure 90: Top view of baseline aircraft

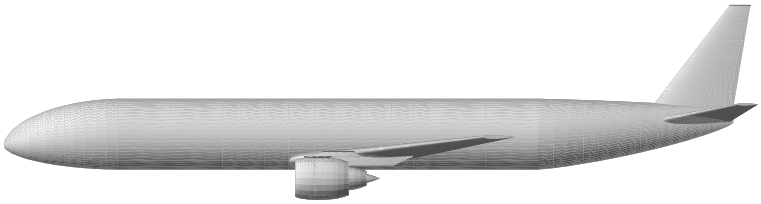

Figure 91: Side view of baseline aircraft

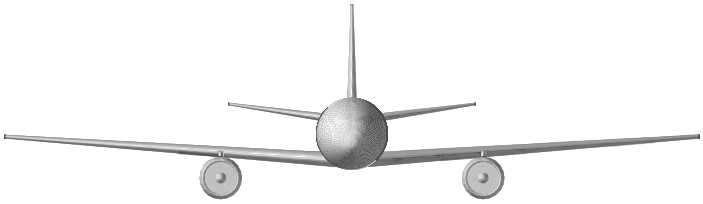

Figure 92: Front view of baseline aircraft

Table 39: Aircraft constraints

Quantity		Value	Units
S_{TO}	$<$	6000	ft
S_{LDG}	$<$	4500	ft
$P_{s,sl}$	$>$	3000	ft/min
$P_{s,cr}$	$>$	300	ft/min

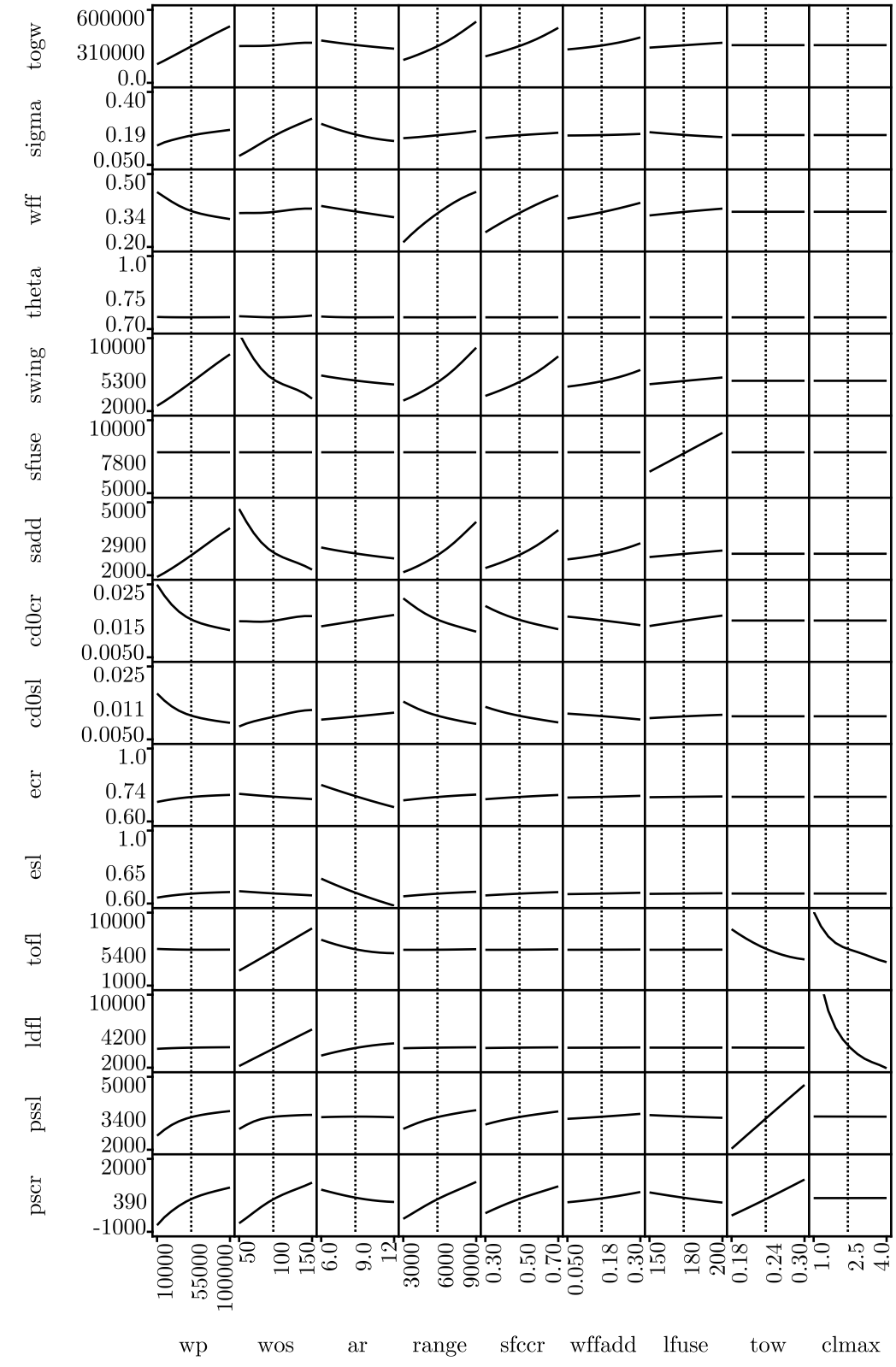

Figure 93: System explorer view of the aircraft design space

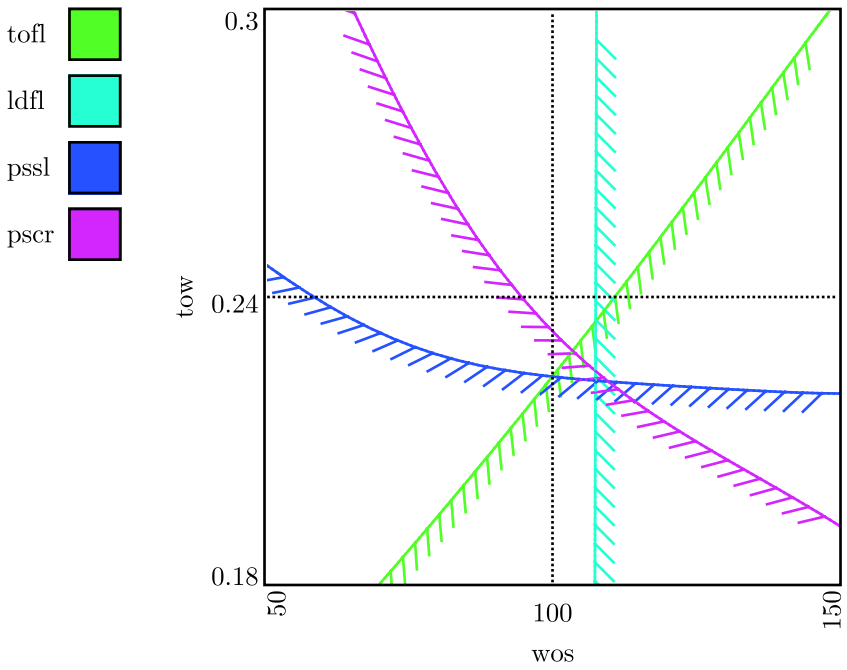

Figure 94: Aircraft constraints

interest and constraint settings representing the aircraft design, and therefore the design and size of the example vehicle, will not be changed. Of course, the environment is also capable of supporting more traditional decisions in design as demonstrated in Chapter 8. This scenario serves as an experiment to test the effectiveness of the fidelity trade environment, in support of testing the research hypotheses.

In this scenario, a designer is at the start of a complex systems design process. The designer has built a system model based upon legacy codes. The designer is interested in accomplishing three major goals. First, the designer wants to choose a baseline design for the vehicle with an understanding of the impact of error on that choice. Second, the designer wants to calculate a preliminary estimate of the size and performance of the baseline design with some understanding of the accuracy of those estimates. Third, the designer wants to make decisions about the required analysis fidelity going forward in the design process. These fidelity decisions will guide investment and development in analysis tools. The fidelity trade environment should enable the designer to accomplish all of these goals.

The first step in the error management exercise is to introduce some representative

sources of error. For problem simplicity and in order to highlight the differing impact of different error sources, the representative error sources are all introduced with the same magnitude, 5%. Similarly, when a fidelity improvement is made, the error source is reduced to 1%. While these error levels are not representative of the actual error of the methods used in the example problem, they illustrate the utility of fidelity trades without introducing the complexity of making and justifying error estimates for every component. The initial error levels applied are listed in Table 40.

Table 40: Initial error sources

Quantity	Value
$\sigma\ C_{L,max}$	5%
$\varsigma\ e_{cr}$	5%
$\varsigma\ W$	5%
$\varsigma\ C_{D,0\ cr}$	5%
$\sigma\ SFC$	5%
$\varsigma\ C_{D,0\ sl}$	5%
$\varsigma\ e_{sl}$	5%
$\varsigma\ W_f/W$	5%

The introductory propagated error breakdown for the system variables is included as Figure 95. Specific excess power at cruise shows a very large percent error; this is largely due to the relatively small magnitude of the excess power relative to the potential changes due to the error sources.

As a verification experiment, a Monte Carlo approach was used to propagate the error through the complex system. The results of the Monte Carlo analysis were compared with the sensitivity based approach in Figure 96. The blue and red bars represent the results of the sensitivity approach and the Monte Carlo approach respectively. The small error bands at the top of the Monte Carlo bar represent the one sigma error in the result due to the use of a finite sample size.

For the Monte Carlo study, Gaussian probability distributions with zero mean and the specified variance were assumed for each of the error sources. A random sample of 10,000 cases were analyzed, and the complete system was brought to convergence for every case.

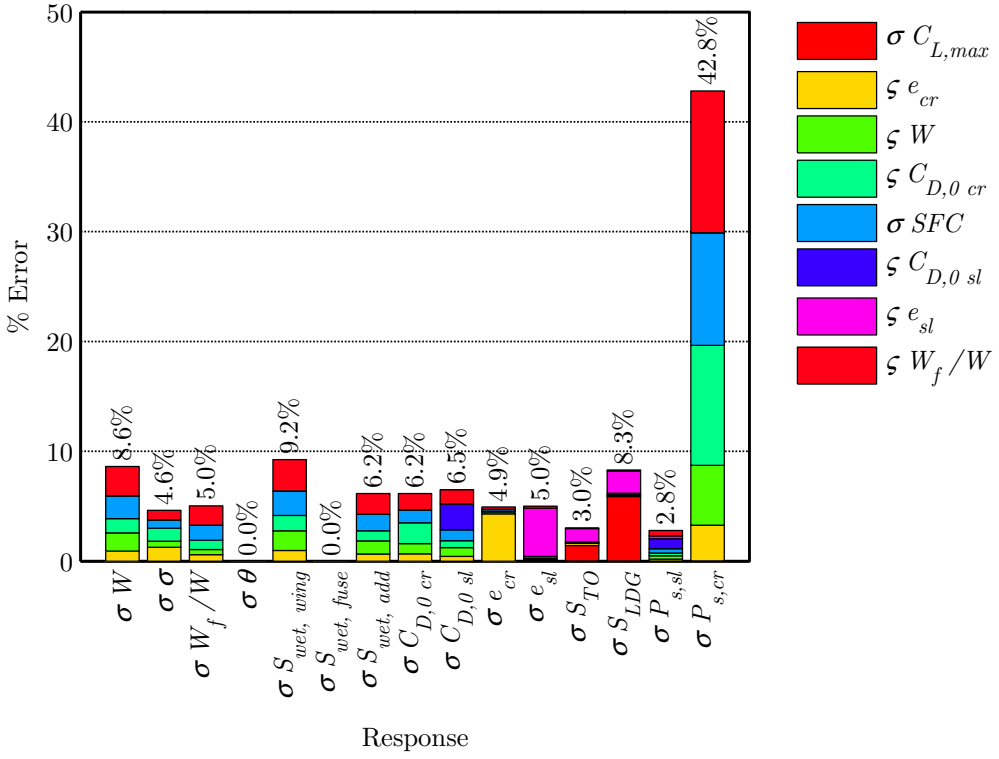

Figure 95: Introductory error breakdown

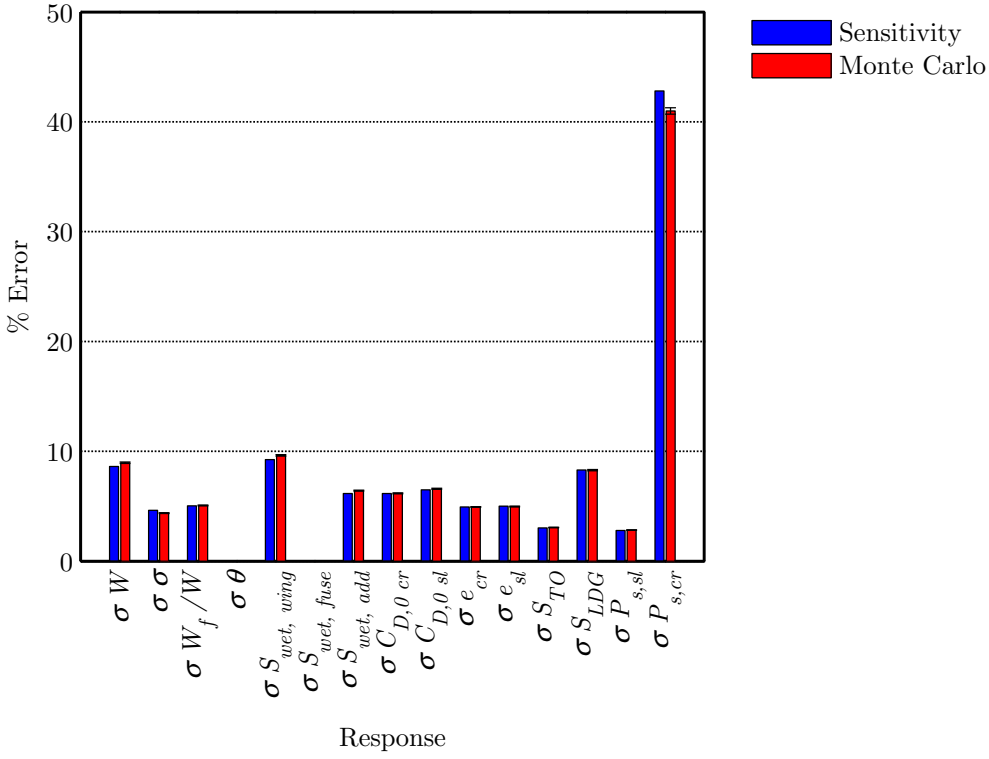

Figure 96: Introductory error verification

In general, the agreement between the two approaches is very good, and the sensitivity approach is verified as an approximate model of the propagation of error through a complex system. The $\sigma P_{s,cr}$ result shows the greatest disparity, clearly beyond that which can be attributed to the finite sample size; this disparity is due to the linearization introduced by the sensitivity approach.

The Monte Carlo verification analysis took about five minutes to complete for a single point in the design space. Each update of the fidelity trade environment would require approximately 500 such analyses. The sensitivity approach allows what would otherwise be a very expensive calculation to be performed in an interactive manner, thereby making the fidelity trade environment possible. The comparative nature of the results displayed by the fidelity trade environment do not require extreme accuracy, and the approximations introduced by the linearization are believed by the author to be acceptable.

The gray band surrounding the system response in Figure 97 represents the system propagated error throughout the design space.

The impact of error on the design space as represented by the constraint diagram is shown in Figure 98. Note that the error on the landing field length constraint has dramatic impact on the choice of the design point of the vehicle. Not only have the constraint locations with the consideration of error changed, but the constraints active in choosing a design point have changed. Before, the design point was constrained by specific excess power at cruise and takeoff field length; afterward, the design point was constrained by specific excess power at cruise and landing field length.

When faced with the constraint diagram depicted in Figure 98, the designer may desire to reduce the error impacting his choice of design point. His first step may be to improve the landing field length estimation. Referring to Figure 95, we note that the primary source of error for landing field length is the error in maximum lift coefficient. The designer may then choose to perform a wind tunnel test to reduce the error on this quantity. The error levels for the system including reduced $C_{L,max}$ error for landing are listed below in Table 41.

The error breakdown corresponding to the system with reduced $C_{L,max}$ error is shown in Figure 99. Note that while the error in takeoff and landing distance were significantly

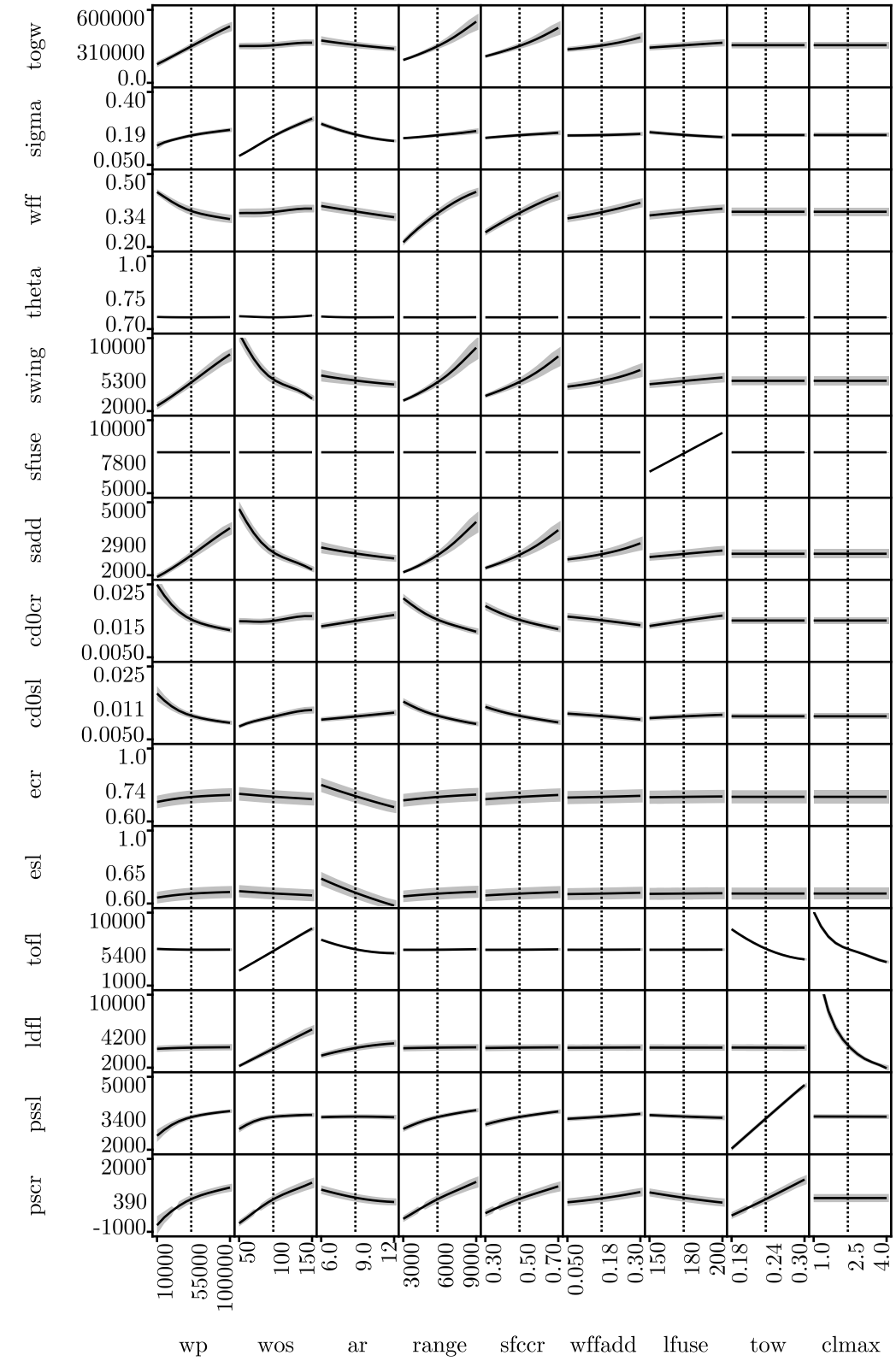

Figure 97: Aircraft design space with introductory error

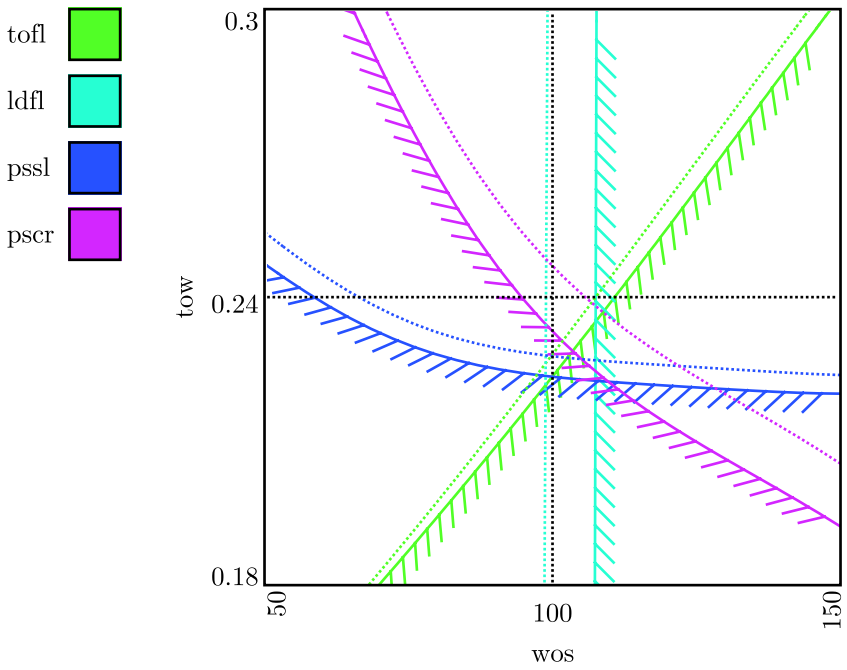

Figure 98: Aircraft constraints with introductory error

Table 41: Reduced error for landing

Quantity	Value
$\sigma\ C_{L,max}$	1%
$\varsigma\ e_{cr}$	5%
$\varsigma\ W$	5%
$\varsigma\ C_{D,0\ cr}$	5%
$\sigma\ SFC$	5%
$\varsigma\ C_{D,0\ sl}$	5%
$\varsigma\ e_{sl}$	5%
$\varsigma\ W_f/W$	5%

reduced, no other quantities were impacted. Similarly, the system explorer with the error distribution throughout the design space is shown in Figure 100; in this figure, the gray bands surrounding the takeoff and landing distance responses in Figure 97 have virtually vanished.

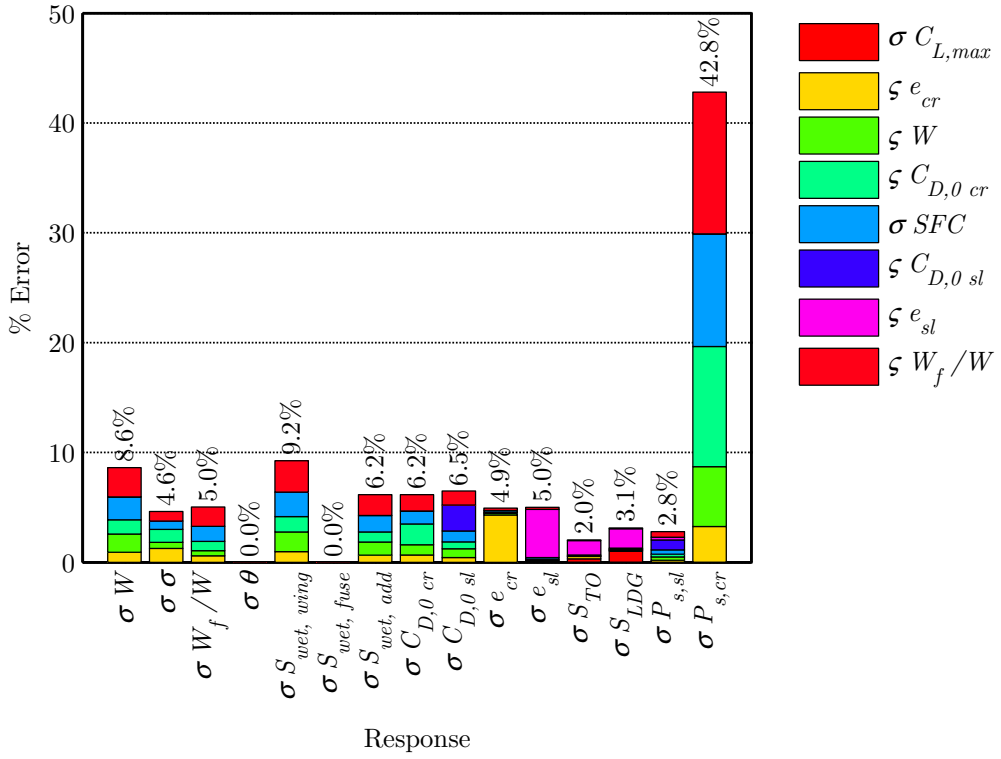

Figure 99: Error breakdown with reduced landing error

Figure 101 demonstrates the effectiveness of reducing the error in $C_{L,max}$ on the design point selection. While the active constraints have still changed, the impact of error on choosing the design wing loading has been minimized.

Examination of Figure 101 demonstrates that the impact of error on the design point is now primarily due to error in specific excess power at cruise. Reducing this error is now the designer's priority. The error breakdown shown in Figure 99 indicates which error sources have the greatest impact on cruise specific excess power. The designer may then choose to reduce the error in some of these quantities as listed in Table 42. The cruise drag coefficient estimate $C_{D,0\ cr}$ could be improved through a wind tunnel test, while an

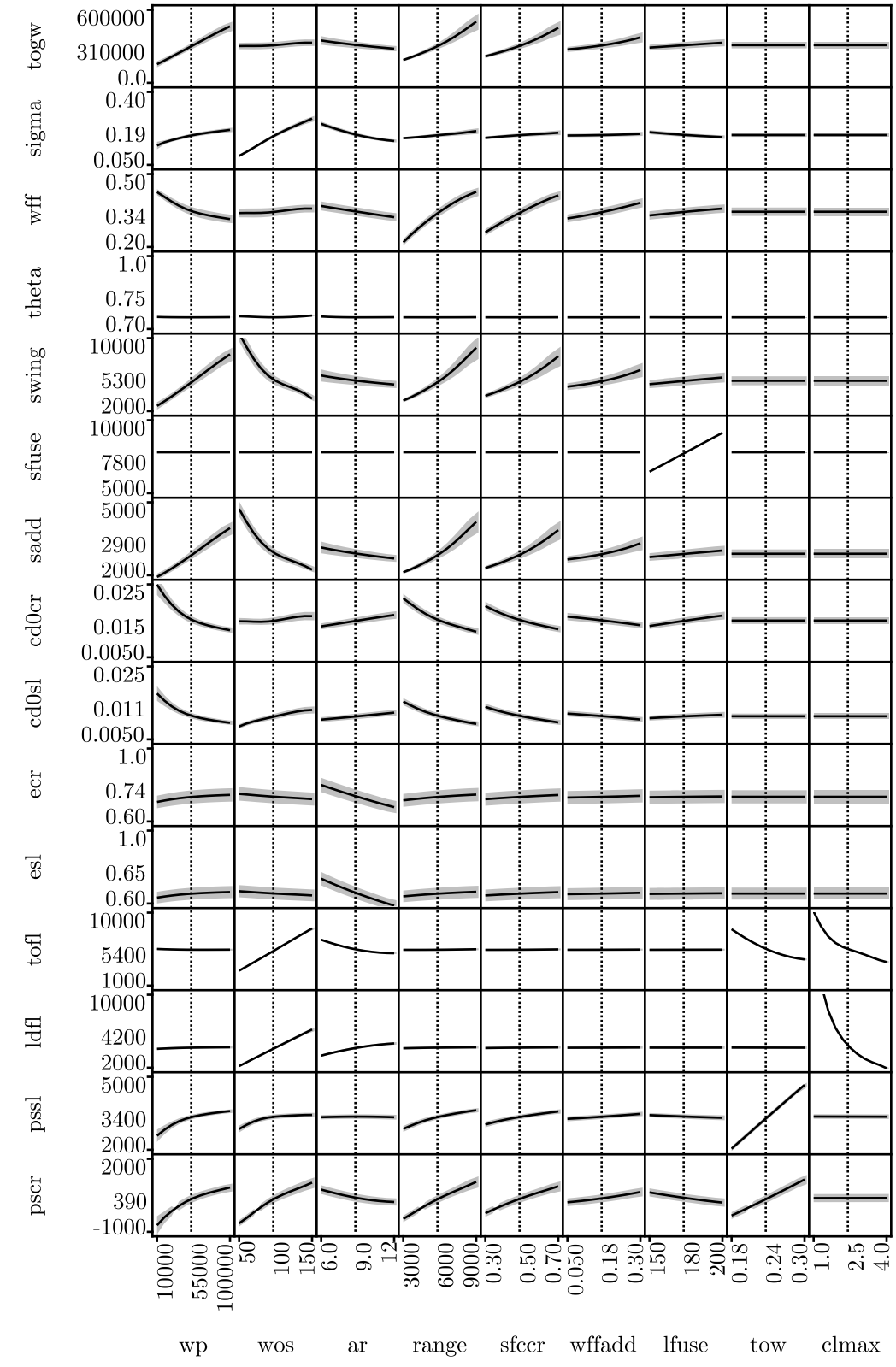

Figure 100: Aircraft design space with reduced landing error

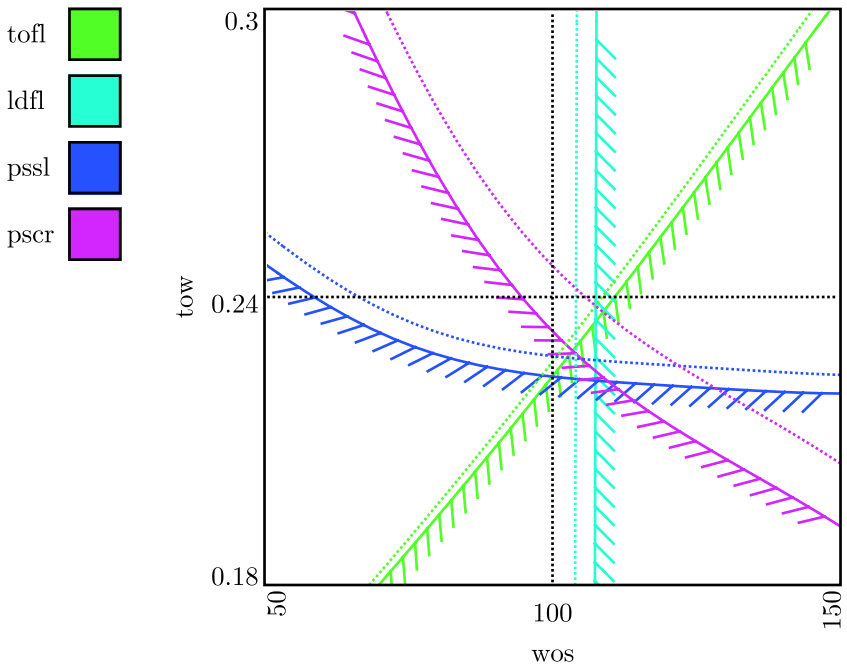

Figure 101: Aircraft constraints with reduced landing error

improved estimate of the cruise specific fuel consumption SFC could be requested from the engine manufacturer. An improved mission model could improve the fuel burn calculation W_f/W.

Table 42: Reduced error for cruise

Quantity	Value
$\sigma\ C_{L,max}$	1%
$\varsigma\ e_{cr}$	5%
$\varsigma\ W$	5%
$\varsigma\ C_{D,0\ cr}$	1%
$\sigma\ SFC$	1%
$\varsigma\ C_{D,0\ sl}$	5%
$\varsigma\ e_{sl}$	5%
$\varsigma\ W_f/W$	1%

The error breakdown corresponding to the system for reduced cruise error is shown in Figure 102. While still showing the largest percent error, error in cruise excess power has been reduced enough to warrant changing the scale of the error breakdown. The corresponding system explorer with the error distribution throughout the design space is shown

in Figure 103; in this figure, the gray bands surrounding most responses in Figures 100 and 97 have been significantly reduced.

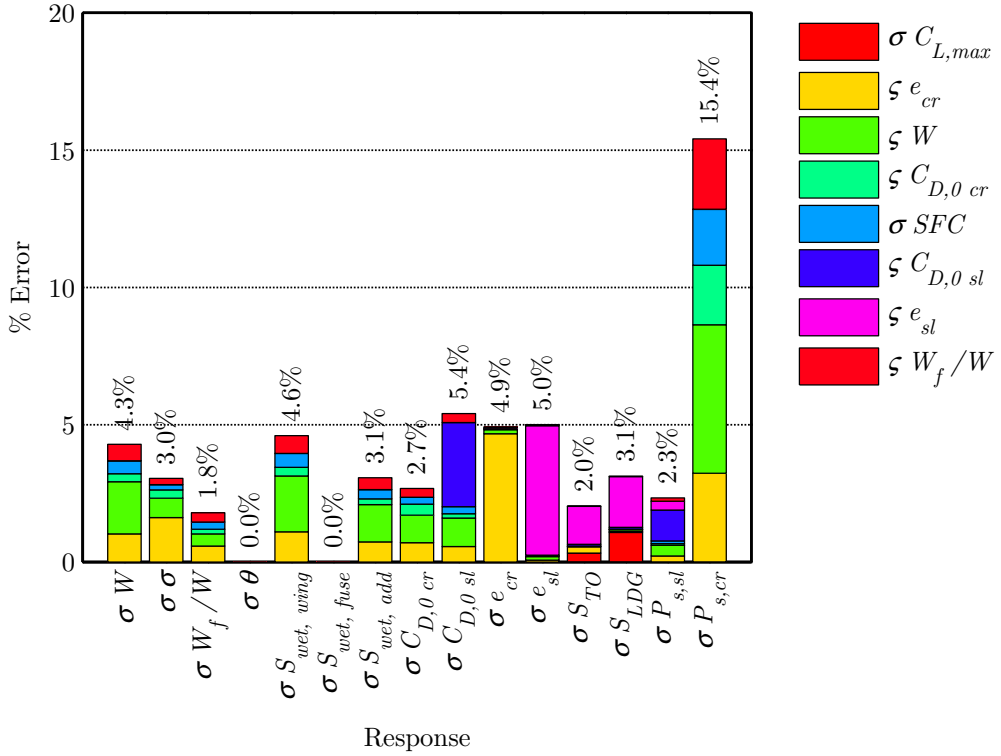

Figure 102: Error breakdown with reduced cruise error

Figure 104 demonstrates the impact of reducing cruise error on the design point selection. As intended, the impact of error on choosing the design thrust loading has been minimized.

Interpretation of Figure 104 may show that the level of error impacting the design point is acceptable to the designer. Once a design is selected, the primary purpose of an aircraft systems study is to estimate the size of the vehicle. The takeoff gross weight of an aircraft is a primary driver of the manufacturing, operating, maintenance, and other costs of the vehicle. In the absence of further analysis, minimization of aircraft weight tends to minimize cost. Examination of the error breakdown shown in Figure 102 shows that the biggest remaining contributor to error in takeoff gross weight is the weight calculation itself. The designer may then choose to reduce the error in the weight calculation as listed in Table 43.

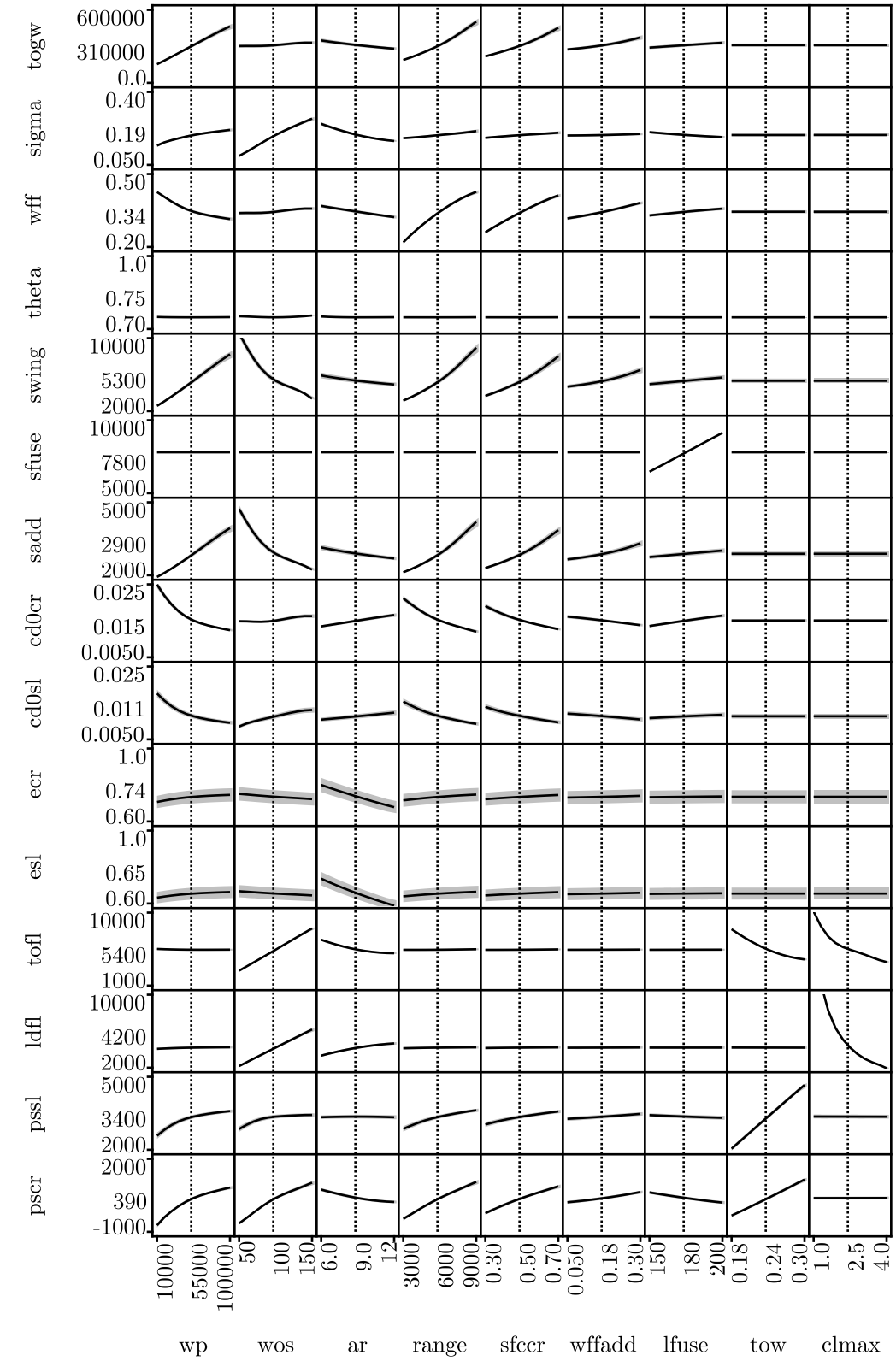

Figure 103: Aircraft design space with reduced cruise error

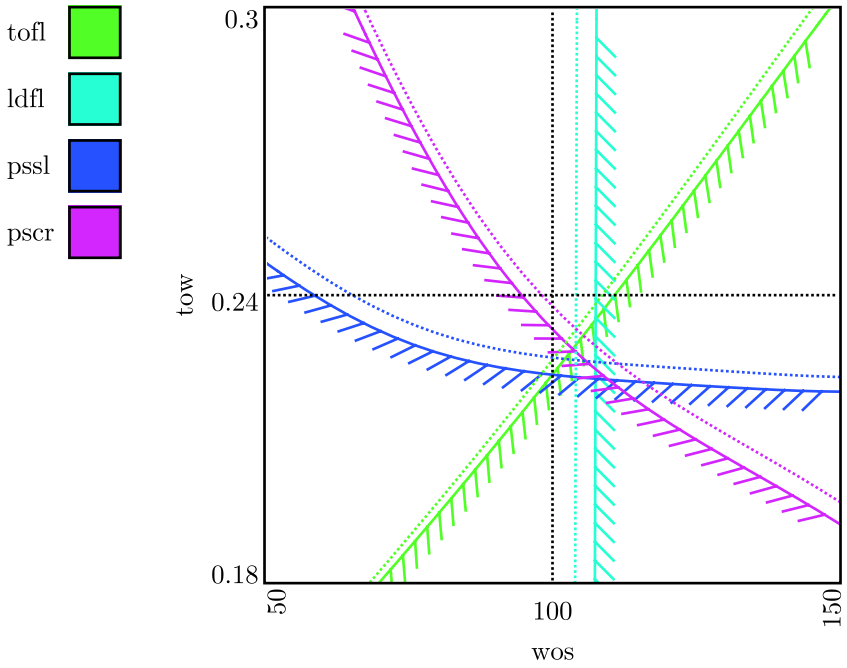

Figure 104: Aircraft constraints with reduced cruise error

Table 43: Reduced error for weight

Quantity	Value
$\sigma\ C_{L,max}$	1%
$\varsigma\ e_{cr}$	5%
$\varsigma\ W$	1%
$\varsigma\ C_{D,0\ cr}$	1%
$\sigma\ SFC$	1%
$\varsigma\ C_{D,0\ sl}$	5%
$\varsigma\ e_{sl}$	5%
$\varsigma\ W_f/W$	1%

The error breakdown corresponding to the system with reduced weight calculation error is shown in Figure 105. As intended, the error in the takeoff gross weight estimation has been significantly reduced from 4.3% to 2.5%. The corresponding system explorer with the error distribution throughout the design space is shown in Figure 106. Similarly, the corresponding constraint diagram is shown in Figure 107.

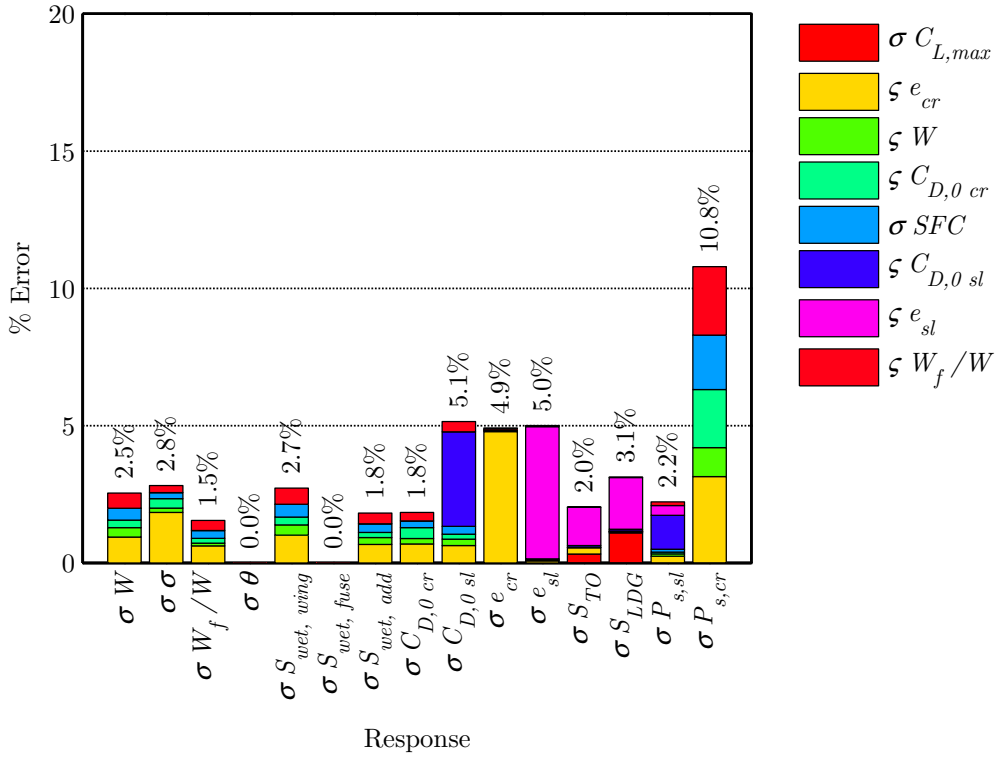

Figure 105: Error breakdown with reduced weight error

In this scenario, the designer was faced with a complex system incorporating a variety of error sources. His first actions were to reduce the sources of error contributing to uncertainty in the choice of the design point. Once a design point could be chosen with confidence, his next action was to reduce the sources of error contributing to the most important system level metrics. In so doing, some significant sources were allowed to remain, being quantitatively determined to be unimportant to the decision making process undertaken by the designer.

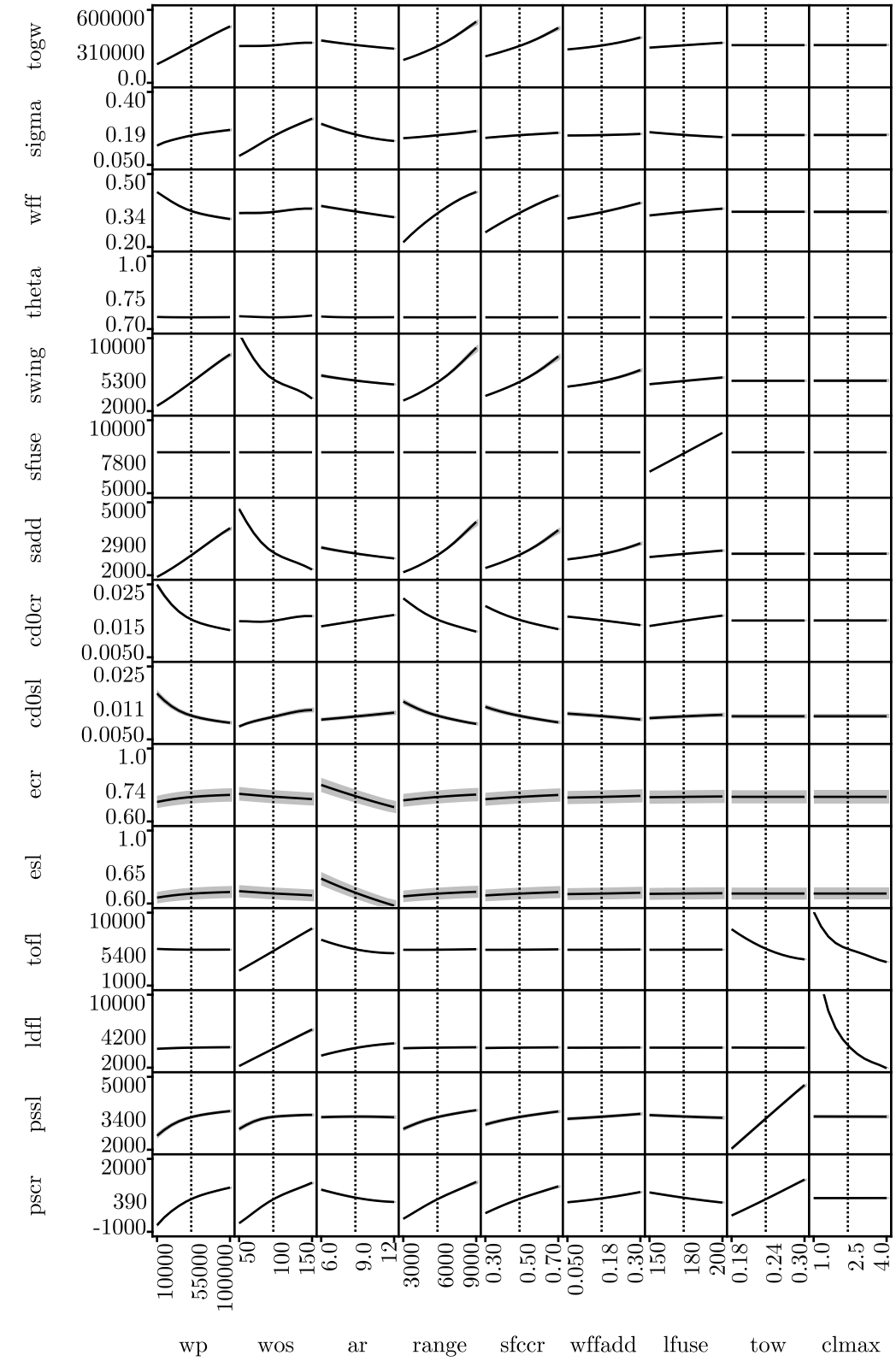

Figure 106: Aircraft design space with reduced weight error

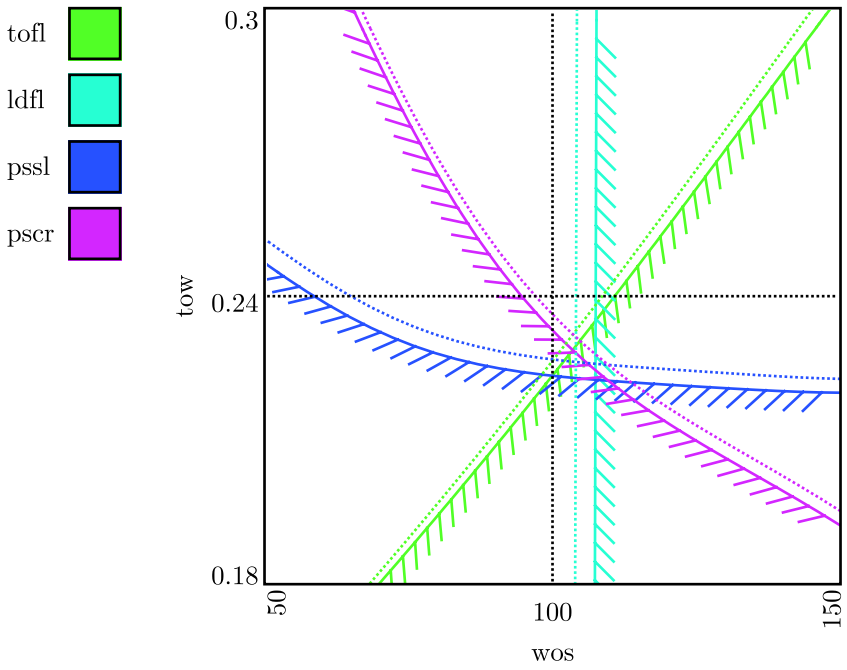

Figure 107: Aircraft constraints with reduced weight error

CHAPTER XIII

MONOLITHIC SYSTEM CASE STUDY

The second primary example presented in this thesis demonstrates an alternate approach to modeling the propagation of error through a complex system. In this example, a single monolithic program capable of modeling the entire complex system is used. This program has been written with a series of *k-factors* which may be used to adjust the result of internal calculations. In this manner, one of the primary advantages of decomposing the system is attained while still allowing the monolithic program to assume the burden of convergence and consistency of the system solution.

From the perspective of the fidelity trade environment, the system now consists of a single component. This component has behavior significantly more complex than the components of the previous example. Because information transfer between modules is handled internally, a greater volume of information may be easily conveyed without any impact on the system perspective. Of course, it is also possible to investigate a decomposed complex system built of monolithic system components. Such an approach would enable modeling extremely complex systems without an explosive growth in complexity.

This case study experiment serves as a further test of the effectiveness of the fidelity trade environment in meeting the guiding research hypotheses.

13.1 System Model

The monolithic system model used in this example was a FLOPS [82] model of a representative twin-engine 300 passenger transport aircraft; the baseline model was provided by Dr. Kirby of the Georgia Institute of Technology. FLOPS is a comprehensive aircraft sizing and synthesis program which includes modules for modeling the weights, aerodynamics, mission, and performance of aircraft. A limited number of inputs, listed in Table 44 were selected for the purposes of this study.

Table 44: Flops task inputs

Quantity	Name	Min.	Max.	Units	Description
W_{eng}	weng	0	–	lbf	Engine weight.
kW_{wing}	kww	0	–	–	Wing weight factor.
kW_{ht}	kwht	0	–	–	Horizontal tail weight factor.
kW_{fuse}	kwf	0	–	–	Fuselage weight factor.
kW_{vt}	kwvt	0	–	–	Vertical tail weight factor.
kW_{gear}	kwg	0	–	–	Landing gear weight factor.
kW_{sc}	kwsc	0	–	–	Surface controls weight factor.
kW_{furn}	kwfu	0	–	–	Furnishings weight factor.
kW_{fsys}	kwfs	0	–	–	Fuel system weight factor.
S	sw	0	–	ft^2	Wing area.
R	range	0	–	nmi	Design range.
T/W	tow	0	–	–	Thrust to weight ratio.
$C_{L,max,to}$	cltx	0	–	–	Maximum takeoff lift coefficient.
$C_{L,max,ld}$	cllx	0	–	–	Maximum landing lift coefficient.
$kSFC$	ksfc	0	–	–	Specific fuel consumption factor.
$kC_{D,0}$	kcd0	0	–	–	Zero-lift drag coefficient factor.
$kC_{D,i}$	kcdi	0	–	–	Induced drag coefficient factor.
$kC_{D,to}$	kcdt	0	–	–	Takeoff drag coefficient factor.
$kC_{D,ld}$	kcdl	0	–	–	Landing drag coefficient factor.

Many of the inputs listed in Table 44 are adjustment factors which scale the results of intermediate calculations made by FLOPS. For example, $kC_{D,0}$ provides an adjustment factor for the estimate of $C_{D,0}$ made by the internal aerodynamics module in FLOPS; setting $kC_{D,0}$ to 0.95 would result in a 5% reduction in $C_{D,0}$.

The k-factors are subject to a variety of interpretations. They may be used as technology factors to simulate the impact of a hypothetical technology. They may be used as calibration factors to correct internal calculations to match a validation case; the use of k-factors in the propagation of error follows from this interpretation. The k-factors can be used in the fidelity trade environment to investigate the propagation of error through a system while simultaneously investigating the impact of technologies or the calibration of the model.

It is important to note that when an error is applied to a quantity via a k-factor in the fidelity trade environment, the error applies to the quantity, not to any adjustments made by changing the k-factor setting. For example, a 3% error applied via $kC_{D,0}$ refers to an error in the $C_{D,0}$ calculation, not to a 3% error in the 5% benefit modeled by setting $kC_{D,0}$

to 0.95.

Some responses of interest were selected from the comprehensive outputs produced by FLOPS. These responses were listed in Table 45.

Table 45: Flops task outputs

Quantity	Name	Units	Description
S_{LDG}	ldfl	ft	Landing field length.
S_{TO}	tofl	ft	Takeoff field length.
W	togw	lbf	Takeoff gross weight.
W_f	wfuel	lbf	Fuel weight.
$P_{s,toc}$	pstoc	ft/min	Excess power at top of climb.
$P_{s,sl}$	psi	ft/min	Initial excess power.
$P_{s,OEI}$	psoei	ft/min	OEI initial excess power.

Because the monolithic system analysis tool is the only component under investigation by the fidelity trade environment, the component-level inputs are by definition also system-level inputs. The input ranges used to generate the component metamodel are therefore also the ranges of interest for the system study. The input ranges and baseline used for this study were listed in Table 46.

13.2 Metamodel Behavior

A training data set was created by a series of ten 50-point Latin Hypercube samples spanning the ranges listed in Table 46. Separate Latin Hypercube samples were used instead of a single 500-point sample to maintain the domain spanning properties of the sample when a subset of the data was used. All of the cases in which FLOPS crashed were discarded and not replaced.

The first 100 cases resulted in 94 data points which did not crash. These 94 points were used to train a Gaussian process metamodel. As a point of reference, a least squares regression would require at least 381 sample points to fit a typical second-order response surface for a problem with 19 input variables. In addition, 1000 random points were run to generate a test data set; the crashed cases were thrown out, leaving 947 points. The test points were used to calculate the metamodel quality metrics described in Section 9.3. The

Table 46: System level variable ranges and settings

Quantity	Min.	Max.	Baseline	Units
W_{eng}	15000	25000	20000	lbf
kW_{wing}	0.75	1.25	1.0	–
kW_{ht}	0.75	1.25	1.0	–
kW_{fuse}	0.75	1.25	1.0	–
kW_{vt}	0.75	1.25	1.0	–
kW_{gear}	0.75	1.25	1.0	–
kW_{sc}	0.75	1.25	1.0	–
kW_{furn}	0.75	1.25	1.0	–
kW_{fsys}	0.75	1.25	1.0	–
S	3000	7000	5000	ft^2
R	6000	10000	8000	nmi
T/W	0.25	0.35	0.3	–
$C_{L,max,to}$	1.5	4.0	2.75	–
$C_{L,max,ld}$	1.5	4.0	2.75	–
$kSFC$	0.75	1.25	1.0	–
$kC_{D,0}$	0.75	1.25	1.0	–
$kC_{D,i}$	0.75	1.25	1.0	–
$kC_{D,to}$	0.75	1.25	1.0	–
$kC_{D,ld}$	0.75	1.25	1.0	–

resulting quality metrics for the metamodels were listed in Table 47.

Table 47: 94-Point metamodel characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
S_{LDG}	2.97%	3.28%	0.988
S_{TO}	11.80%	12.85%	0.903
W	1.49%	1.54%	0.992
W_f	2.97%	2.79%	0.993
$P_{s,toc}$	248.7%	10.02%	0.972
$P_{s,sl}$	6.48%	3.31%	0.985
$P_{s,OEI}$	119.5%	14.37%	0.983

The actual versus predicted responses for the test points of the metamodels were plotted as Figure 108.

Figure 108 and Table 47 reveal that the FLOPS metamodel is quite good for being based on only 94 training points. The S_{TO} metamodel shows the worst fit of the responses, while the $P_{s,sl}$ and $P_{s,OEI}$ responses show the elevated RMSE_{x_i} values expected from small

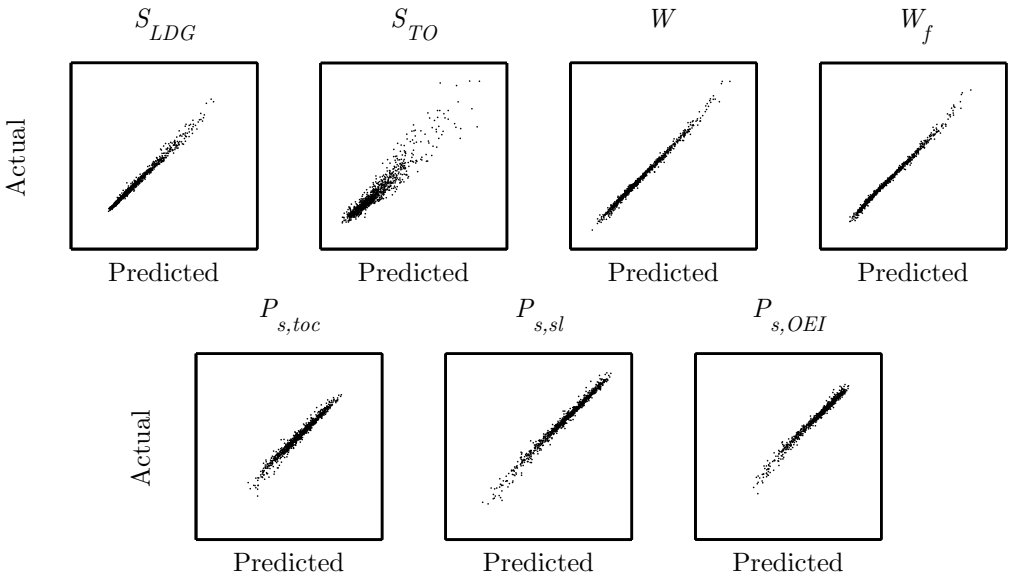

Figure 108: 94-Point metamodel response

responses.

Further examination of the residuals in the S_{TO} prediction reveal that the prediction is quite good for reasonable takeoff distances, with error growing when $S_{TO} > 10000ft$. Poor prediction in this situation is primarily due to a lack of training data in the corner of the design space leading to very long takeoff field lengths. These points are typified by low T/W and S and by large $kC_{D,to}$. These were also the points most likely to crash and be discarded from the training data. While an increase in the size of the training data set should help improve the prediction of this response, the difficulty in obtaining training data in that corner of the design space will persist. This regional deficit is not of great concern because it only occurs in truly undesirable portions of the design space and because it is not sufficient to make those regions appear desirable.

The training data set was expanded to the first 300 cases, which resulted in 285 cases which did not crash. These 285 points were used with the hyperparameters from the 94 point metamodel. The same 947 point test data set was used to calculate the metamodel quality metrics described in Section 9.3. The resulting quality metrics for the metamodels were listed in Table 48.

The actual versus predicted responses for the test points of the metamodels were plotted

Table 48: 285-Point metamodel characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
S_{LDG}	1.91%	2.21%	0.994
S_{TO}	6.87%	8.18%	0.958
W	0.60%	0.70%	0.998
W_f	1.17%	1.37%	0.998
$P_{s,toc}$	58.43%	6.48%	0.988
$P_{s,sl}$	8.20%	1.64%	0.996
$P_{s,OEI}$	54.4%	7.13%	0.995

as Figure 109.

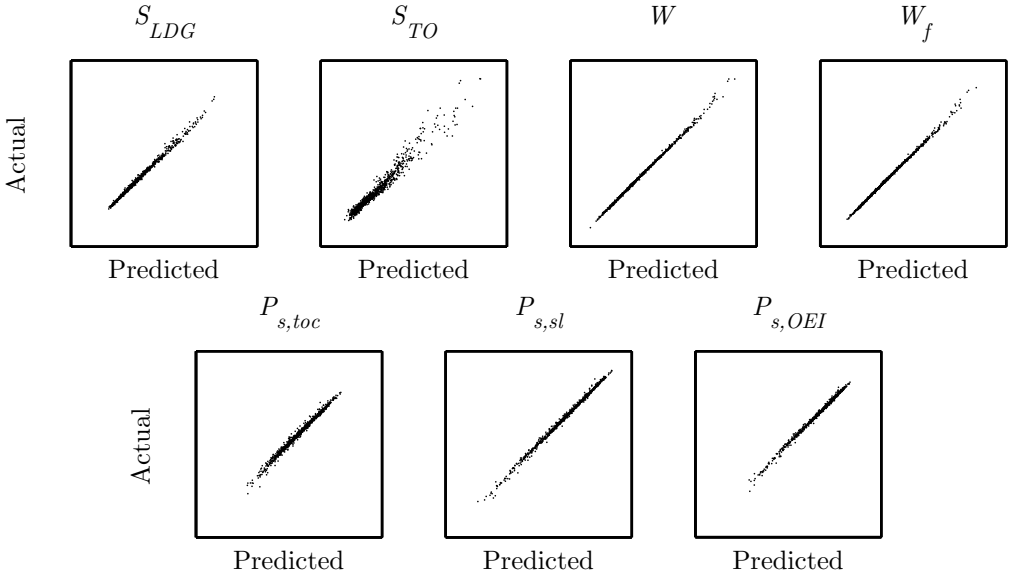

Figure 109: 285-Point metamodel response

Figure 109 and Table 48 reveal significant improvement in the 285 point metamodel over the 94 point metamodel. The S_{TO} metamodel continues to show the worst fit of the responses, while the $P_{s,sl}$ and $P_{s,OEI}$ responses show the elevated RMSE_{x_i} values expected from small responses.

The training data set was completely expanded to contain all 500 cases, which resulted in 471 cases which did not crash. These 471 points were used with the hyperparameters from the 94 point metamodel. The same 947 point test data set was used to calculate the metamodel quality metrics described in Section 9.3. The resulting quality metrics for the

metamodels were listed in Table 49.

Table 49: 285-Point metamodel characteristics

Quantity	RMSE_{x_i}	$\text{RMSE}_{\bar{x}}$	R^2
S_{LDG}	1.54%	1.84%	0.996
S_{TO}	6.44%	7.54%	0.964
W	0.47%	0.56%	0.99898
W_f	0.95%	1.15%	0.9988
$P_{s,toc}$	95.10%	5.29%	0.972
$P_{s,sl}$	3.35%	1.15%	0.998
$P_{s,OEI}$	27.91%	5.12%	0.99795

The actual versus predicted responses for the test points of the metamodels were plotted as Figure 110.

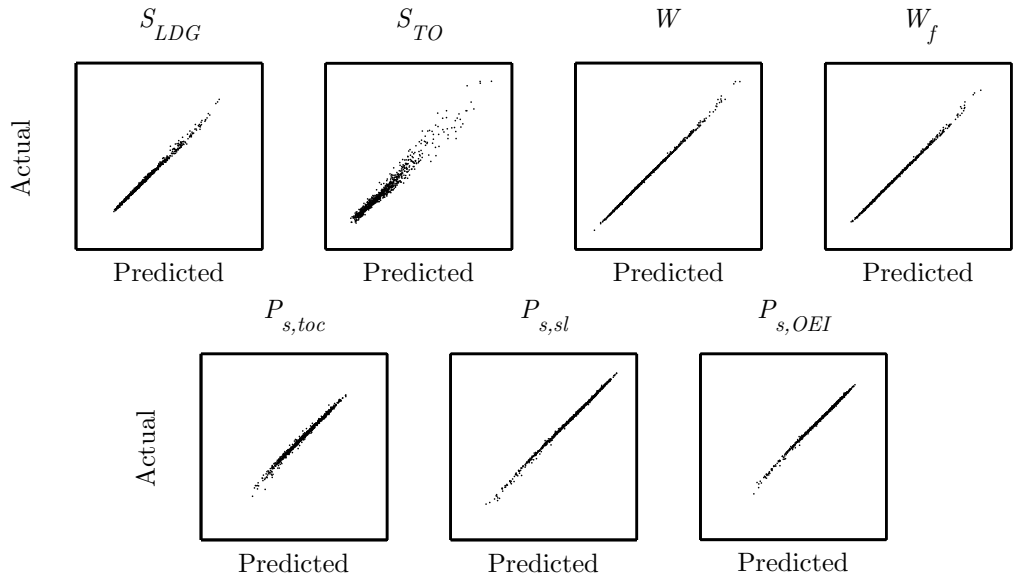

Figure 110: 471-Point metamodel response

Figure 110 and Table 48 reveal significant improvement in the 471 point metamodel over the 285 and 94 point metamodels. The S_{TO} metamodel continues to show the worst fit of the responses, while the $P_{s,sl}$ and $P_{s,OEI}$ responses show the elevated RMSE_{x_i} values expected from small responses.

This demonstration has shown the capability of a Gaussian process metamodel to satisfactorily capture complex behavior over a wide range of responses using a very small training

data set.

13.3 System Response

The system response at the baseline point of interest was listed in Table 50. For the purposes of this study, representative performance constraints were placed on four of the system responses as listed in Table 51.

Table 50: Sized vehicle characteristics

Quantity	Value	Units
S_{LDG}	5800	ft
S_{TO}	5000	ft
W	590000	lbf
W_f	244000	lbf
$P_{s,toc}$	920	ft/min
$P_{s,sl}$	3070	ft/min
$P_{s,OEI}$	860	ft/min

Table 51: Aircraft constraints

Quantity		Value	Units
S_{LDG}	$<$	6500	ft
S_{TO}	$<$	6500	ft
W	$<$	625000	lbf
$P_{s,OEI}$	$>$	300	ft/min

In order to investigate the propagation of error through the complex system aircraft model, error sources were introduced on many of the input quantities whether directly, or through the k-factors. The base error levels were uniformly set to 5% as listed in Table 52 to highlight the differential impact of each error source.

The base propagated error breakdown for the system variables is included as Figure 111. One engine out and top of climb specific excess power show inflated error levels; this is largely due to the relatively small magnitude of the excess power relative to the potential changes due to the error sources. Despite the fact that all of the error sources are equal in magnitude, they have widely varying impact on the system response. Furthermore, the

Table 52: Base error sources

Quantity	Value
ςW_{eng}	5%
ςW_{wing}	5%
ςW_{ht}	5%
ςW_{fuse}	5%
ςW_{vt}	5%
ςW_{gear}	5%
ςW_{sc}	5%
ςW_{furn}	5%
ςW_{fsys}	5%
$\varsigma C_{L,max,to}$	5%
$\varsigma C_{L,max,ld}$	5%
ςSFC	5%
$\varsigma C_{D,0}$	5%
$\varsigma C_{D,i}$	5%
$\varsigma C_{D,to}$	5%
$\varsigma C_{D,ld}$	5%

impact of any one error source varies greatly with the response of interest.

In the same manner as the previous example, a Monte Carlo approach was used to conduct a verification experiment propagating the error through the complex system. The results of the Monte Carlo analysis were compared with the sensitivity based approach in Figure 112. The blue and red bars represent the results of the sensitivity approach and the Monte Carlo approach respectively. The small error bands at the top of the Monte Carlo bar represent the one sigma error in the result due to the use of a finite sample size.

The agreement between the two approaches is excellent, and the sensitivity approach is verified as an approximate model of the propagation of error through monolithic this complex system.

The system response throughout the design space was included as Figure 113. The thin gray band surrounding the system response represents the system propagated error throughout the design space.

Comparison of the system response to the engine weight and various weight k-factors in Figure 113 reveal great similarities. The weight of each component impacts the overall system performance in the same way; this is seen through the similarities of the trends for

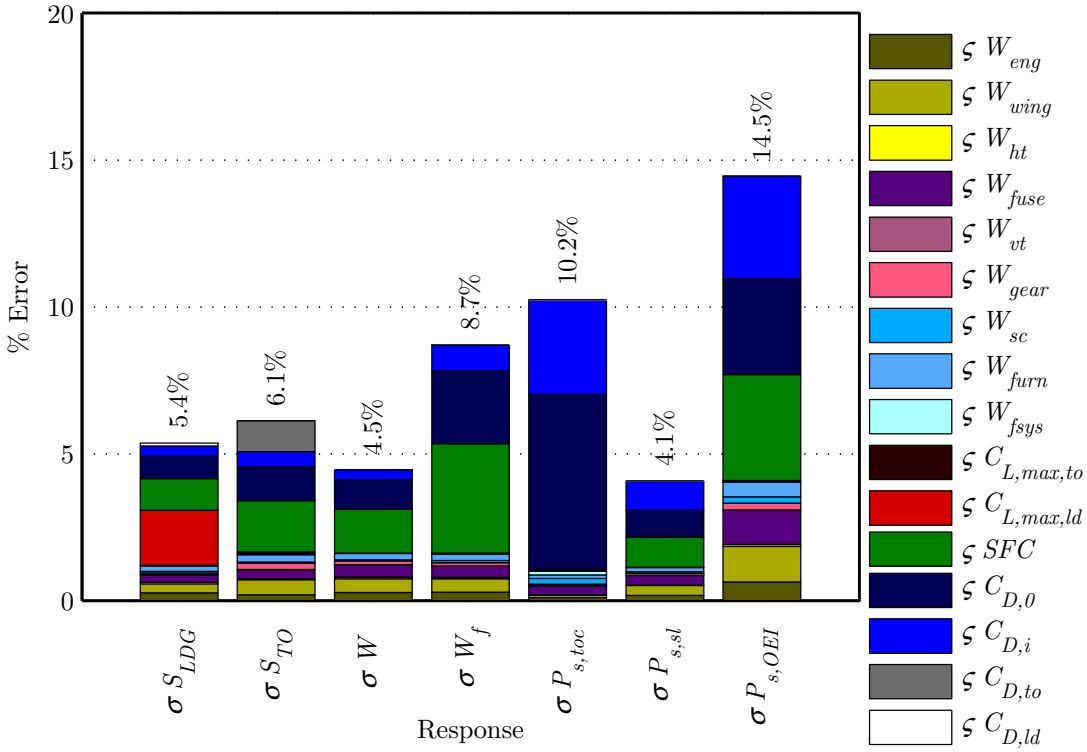

Figure 111: Error breakdown with base error

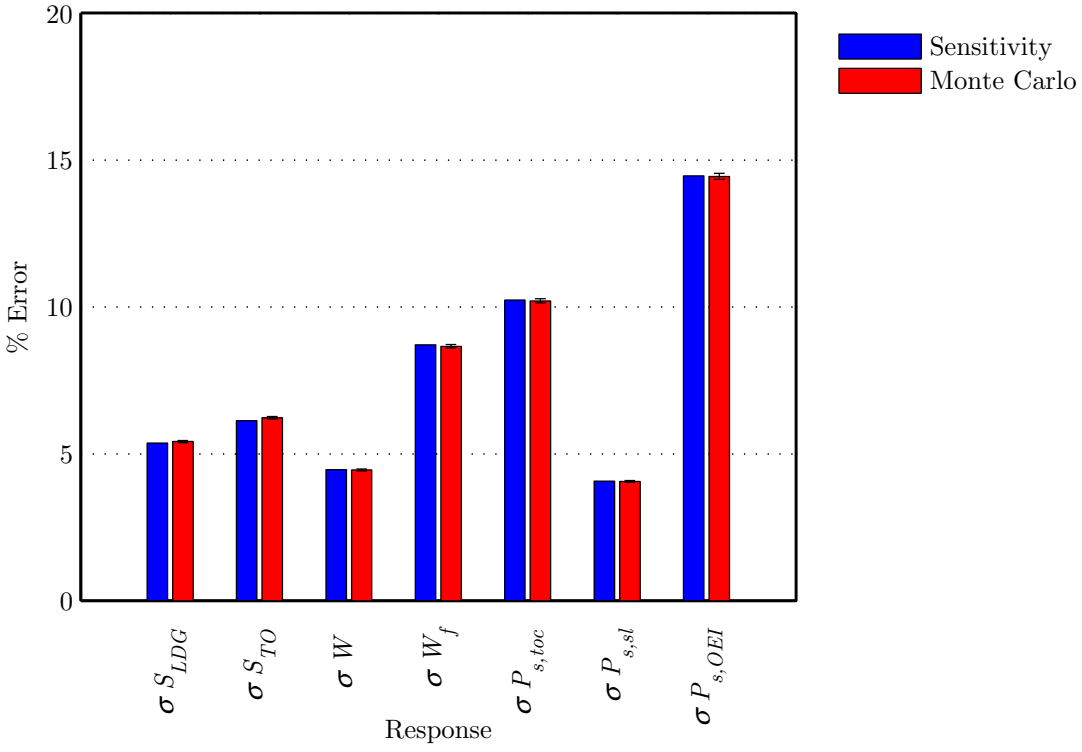

Figure 112: Base error verification

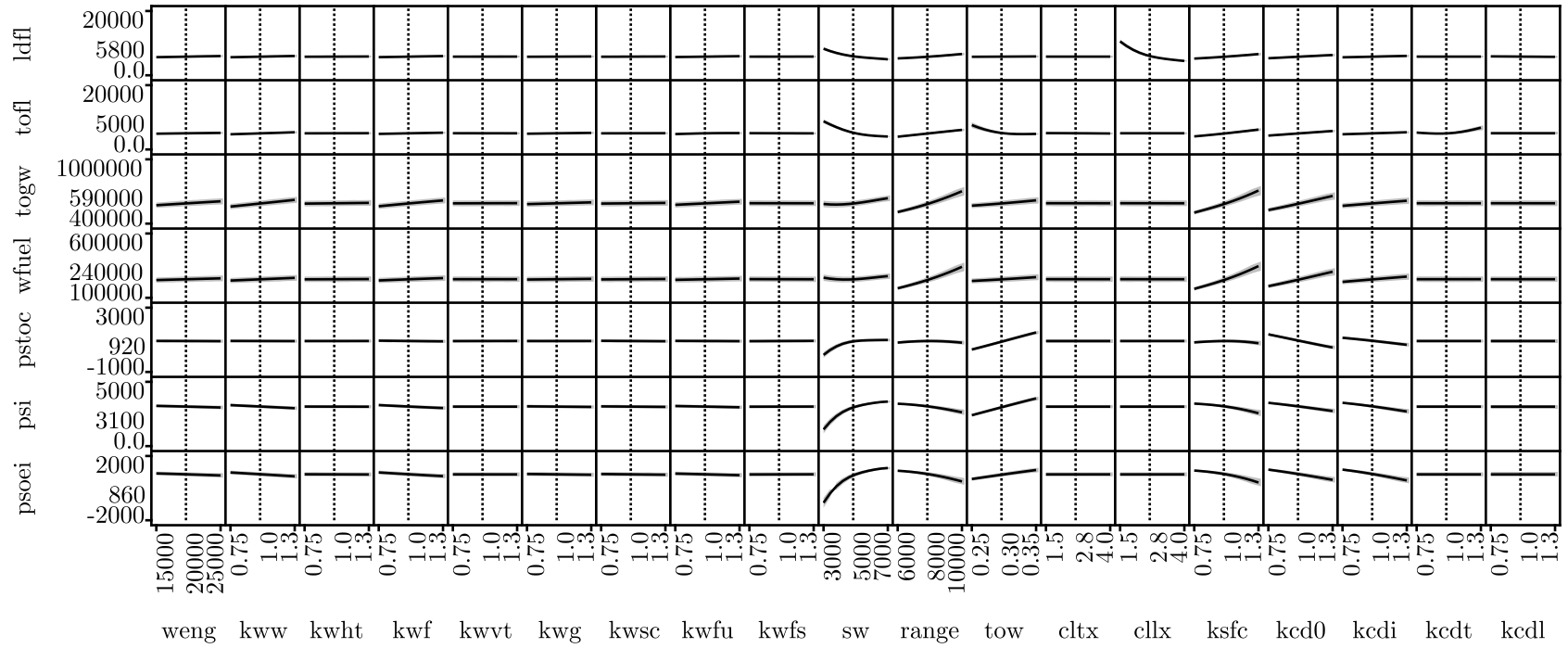

Figure 113: Flops aircraft design space with base error

each input for a given response. However, the magnitude of this impact is directly related to the weight of the component in question; a 5% reduction in fuselage weight will have a greater system impact than a 5% reduction in fuel system weight.

If, instead of a k-factor on the result of a component weight calculation, the k-factor was placed on an intermediate weight calculation (such as maximum stress, root bending moment, etc.), then one would expect the trends resulting from these adjustment factors to take on more varied behavior.

The impact of error on the design space as represented by the constraint diagram is shown in Figure 114. Note that this constraint diagram depicts T/W vs. S instead of W/S. This change dramatically alters the appearance of typical performance constraints and may require special attention. In this figure, smaller and cheaper aircraft, characterized by small wing areas and low thrust loading, appear in the bottom left corner. As expected, this direction of improvement is generally perpendicular to the weight constraint curve.

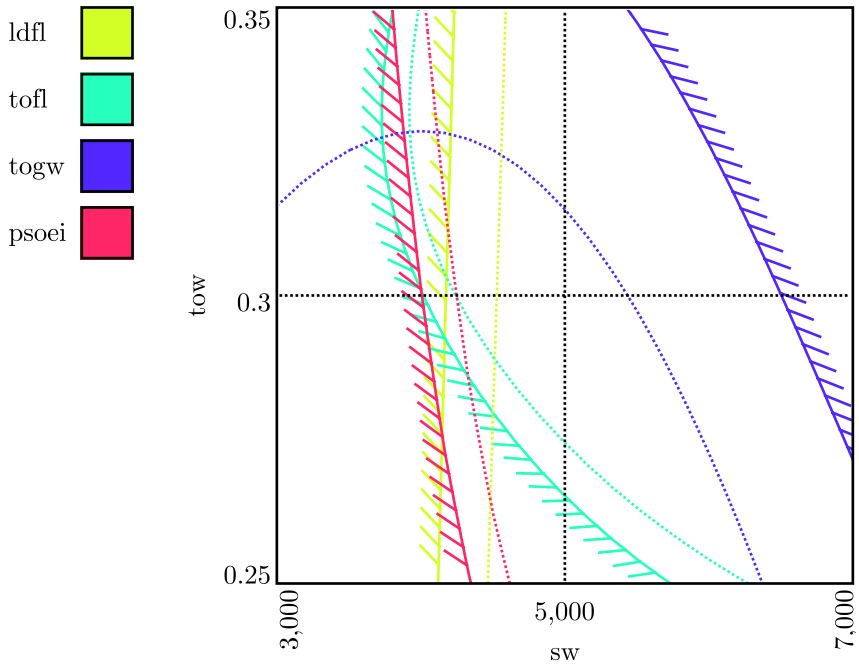

Figure 114: Flops aircraft constraints with base error

With consideration of error, it is not clear from Figure 114 what design point should be selected or what constraints are active at the design point. Revisiting Figure 111 reveals

that while some error sources have significant impact on certain system responses (such as $\varsigma C_{L,max,ld}$ on S_{LDG} and $\varsigma C_{D,to}$ on S_{TO}), three error sources have the greatest impact on all of the metrics; these three are ςSFC, $\varsigma C_{D,0}$, and $\varsigma C_{D,i}$. The three primary error contributors were target for a notional fidelity improvement and their error levels were reduced to 1%, resulting in the error sources listed in Table 53.

Table 53: Reduced error sources

Quantity	Value
ςW_{eng}	5%
ςW_{wing}	5%
ςW_{ht}	5%
ςW_{fuse}	5%
ςW_{vt}	5%
ςW_{gear}	5%
ςW_{sc}	5%
ςW_{furn}	5%
ςW_{fsys}	5%
$\varsigma C_{L,max,to}$	5%
$\varsigma C_{L,max,ld}$	5%
ςSFC	1%
$\varsigma C_{D,0}$	1%
$\varsigma C_{D,i}$	1%
$\varsigma C_{D,to}$	5%
$\varsigma C_{D,ld}$	5%

The reduced propagated error breakdown for the system variables is included as Figure 115; the overall scale of this chart has been changed from Figure 111 to highlight the impact of the smaller contributors to error. One engine out specific excess power still shows the greatest error level, but landing field length now has the second largest error; the largest contributor to error in landing field length is $\varsigma C_{L,max,ld}$.

The error bands surrounding the system responses for the reduced error levels vanished; consequently, no figure depicting the reduced error system response was included. The impact of error on the design space as represented by the constraint diagram is shown in Figure 116.

Reducing the three primary contributors to error had dramatic impact on the role of error in the constraint diagram and on the size of the design space which can be considered

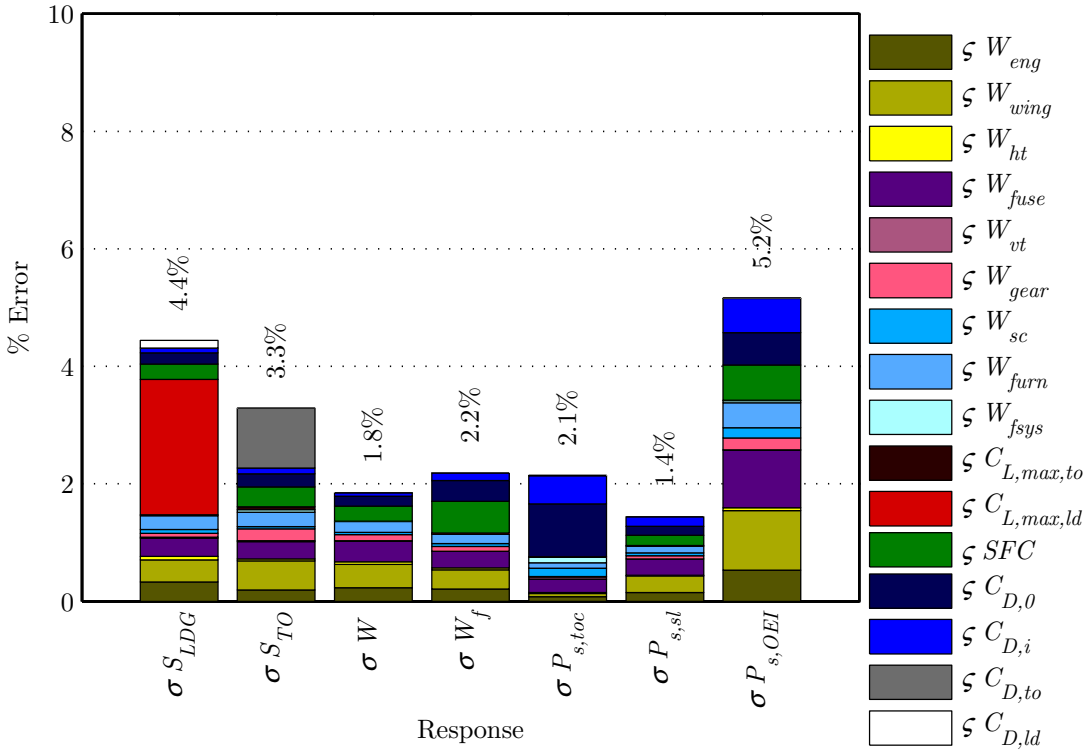

Figure 115: Error breakdown with reduced error

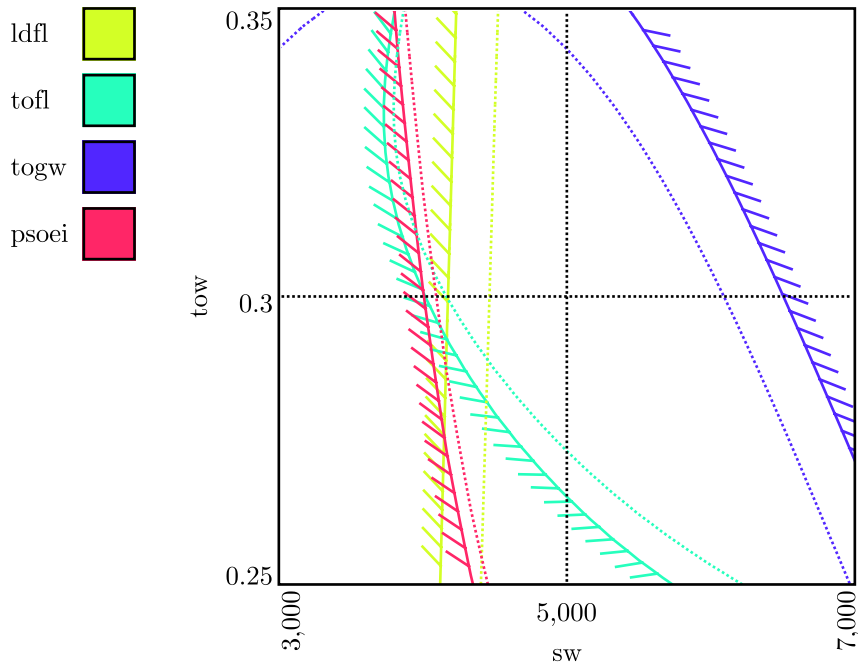

Figure 116: Flops aircraft constraints with reduced error

with confidence. The designer can now be relatively confident that the design point is constrained by S_{LDG} and S_{TO}. Figure 115 reveals that further reductions in the error of these constraints are best obtained by reducing $\varsigma C_{L,max,ld}$ and $\varsigma C_{D,to}$.

CHAPTER XIV

CONCLUSIONS

In Chapter 1, complex systems design was introduced as an emerging field faced with many challenges. One particular challenge is that every step of the systems design process introduces error. Furthermore, the direct validation of complex systems analysis models and design techniques is not possible; instead, a build-up approach to validation must be used where a complex systems analysis is built from trusted component analyses. Unfortunately, error in complex systems can behave in complex and counterintuitive ways such that the build-up approach to systems validation is not sufficient to ensure a valid systems model. At present, there are no techniques available which allow the decision maker to comprehensively consider fidelity and error in a complex systems design process.

These fundamental observations were summarized in the motivating research question given in Chapter 2 and repeated below for clarity.

Research Question: *How can the designer better make fidelity decisions in a complex systems design process?*

In response to this motivating question, a fidelity trade environment was conceived as a decision making tool which combined the impact of error in a complex system with familiar tools for complex systems design. This solution was formally proposed by the hypothesis statement given in Chapter 2 and repeated below.

Hypothesis 1: *If a fidelity trade environment is created then, when it is used in a complex systems design process, it will improve the designer's ability to make fidelity decisions.*

Testing this hypothesis is predicated upon the design and development of the theory and implementation of a fidelity trade environment and upon the development of a representative systems model in the fidelity trade environment. Then an experiment designed to test the hypothesis can be conducted.

Chapter 3 discusses existing systems design techniques and how a fidelity trade environment fits into that landscape. It goes on to outline a set of required enablers for the development of a fidelity trade environment. Three additional hypotheses are presented presenting the need for these enabling technologies.

Hypothesis 2: *The propagation of error through a complex system can be modeled efficiently, including the isolated impact of individual error sources.*

Hypothesis 3: *Metamodels providing for the efficient approximation of arbitrarily complex functions and their derivatives on arbitrarily large domains can be created and integrated into a complex systems design environment.*

Hypothesis 4: *Complex systems design information can be stored in a comprehensive, standardized, and centralized way.*

Chapter 3 concludes by presenting a document roadmap. In this roadmap, the thesis has three primary parts, a part discussing the enabling techniques (Ch. 4-7), a part discussing the specific implementation chosen for the enablers (Ch. 8-10), and a part which conducts experiments to test the fidelity trade environment and conclude the research (Ch. 12-14).

Chapter 4 which goes into depth about the behavior of error in a complex system. This includes the derivation of equations needed to model individual sources of error propagating through a complex system and how they combine. The tools needed for a designer to make decisions concerning error are also conceived and discussed.

Chapter 5 discusses the role of metamodeling in complex systems design, as well as some broad metamodel classifications and criteria for choosing a metamodeling technique for a given problem. Based on these criteria, a Gaussian process metamodel is chosen for the fidelity trade environment.

Chapter 6 explores the creation, storage, and management of information throughout the complex systems design process. The need for a comprehensive information management system in the fidelity trade environment is established. A data architecture capable of representing complex systems data and metadata in accordance with the black box paradigm

used in complex systems design is proposed. Furthermore, this architecture is designed to provide mechanisms for information protection, validation, assurance, and accountability.

In Chapter 7, an overall architecture for the fidelity trade environment is proposed. This architecture will allow the fidelity trade environment to be seamlessly integrated into an existing complex systems design environment. Then in Chapter 7.5, the choices made detailing the implementation of the fidelity trade environment are discussed.

Chapter 8 describes the primary decision making tool of the fidelity trade environment, the system error visualization environment. First, some theory required for creating system models and generating the systems visualization tools which constitute the user interface is discussed. Then the decision making interface is introduced, described, and documented. This includes all displays of information and the means of carrying out decisions presented to the user. Finally, the the decision making interface is explored for a simple example system. The behavior of the example system in response to the decision making controls is demonstrated and explained.

Chapter 9 discusses in-depth the Gaussian process component of the fidelity trade environment. First, the theory supporting Gaussian process metamodels is presented in a pedagogical manner, with all the modifications and implementation decisions particular to this work explained and justified. Then the decision making interface for training and tuning the Gaussian process metamodel is presented. This interface includes unique visualization of a multi-dimensional metamodel with its training data. Finally, the Gaussian process metamodel is explored for a simple example component. This exercise provides intuition to the decision maker regarding the behavior of Gaussian processes.

Chapter 10 details the database schema used in the fidelity trade environment. This schema implements the subset of the data architecture presented in Chapter 6 required to test the hypothesis and prove the utility of a fidelity trade environment. Then Chapter 11 describes and documents the support components required to complete the architecture proposed in Chapter 7. The support components include the experiment designer used to queue metamodel training points for execution, the executor used to run training points in parallel over distributed computing resources, and the setup interface used to maintain the

data stored in the database.

Chapter 12 introduces a simplified transport aircraft model as a complex system model appropriate for testing the research hypothesis. First, the individual disciplinary components contributing to the system model are decomposed, described, and documented in detail. Then the combined system model is presented including a description of the resulting aircraft design, and various standard depictions of the aircraft design space without the consideration of error.

Chapter 13 introduces another transport aircraft model as a complex system model appropriate for testing the research hypothesis. In the first example, the system was decomposed into contributing components, which were assembled in the fidelity trade environment. In this example, a monolithic systems analysis program was modeled in the fidelity trade environment. This program includes many k-factors which allow access inside the program to simulate the effects of error.

Verification experiments were conducted as a test of Hypothesis 2 to demonstrate the correctness of the sensitivity approach to error propagation for each example complex system.

Experiments were conducted throughout the research to demonstrate the quality of the approximation obtained by the Gaussian process metamodel. In addition, a verification experiment was conducted to demonstrate the correctness of the Gaussian process analytical derivative derived in Chapter 9. Together, these experiments test Hypothesis 3.

Sections 12.3 and 13.3 test the main research hypothesis, Hypothesis 1, and implicitly test Hypothesis 4. In them, scenarios are described where a decision maker uses the fidelity trade environment at the beginning of a complex systems design problem. Using the environment, the designer is able to make design decisions while considering error and he is able to make decisions regarding required tool fidelity as the design problem continues. These decisions could not be made in a quantitative manner before the fidelity trade environment was developed. The fidelity trade environment intuitively displayed the information needed to make decisions and also provided mechanisms for the designer to act on his decisions.

This research led to the conception, formulation, and creation of a fidelity trade environment. The fidelity trade environment was used to model two representative complex systems. Successful decision making processes concerning the fidelity of tools in complex systems design studies were demonstrated; these scenarios were not possible without the fidelity trade environment.

As motivation, a series of challenging questions faced by the designer of a complex system were presented in Chapter 2, repeated below for convenience. This research has developed the fidelity trade environment as a decision making tool to help the designer answer questions like: Is the fidelity of a complex systems analysis adequate, or is an improvement needed? If an improvement is needed, how is that improvement best achieved? Where should limited resources be invested for the improvement of fidelity? How does knowledge of the imperfection of a model impact design decisions based on the model? How does this knowledge impact the choice and certainty of the design point? How does it impact the certainty of the performance of a particular design?

The fidelity trade environment has been shown to be an effective tool for helping the designer make important fidelity decisions in complex systems design.

14.1 Contributions

The formulation and development of the fidelity trade environment is the primary contribution of this research. However, this also required several other significant contributions, and many minor contributions, to the field of complex systems design. Some of these secondary contributions have been summarized below. Where possible, references to the main discussion of each contribution have been included.

The fidelity trade environment was integrated seamlessly with an existing design environment (7.2). This includes the system error visualization environment, metamodel, and data repository. This was accomplished by implementing the environment as a middle layer in an existing client server architecture. Consequently, all of the design space exploration, design of experiments, optimization, visualization, etc. capabilities implemented as clients of the architecture immediately benefit from the advances made in this research.

A unique decision making interface combining fidelity and design decisions was created (8.2). This required the development of the equations governing the flow of error through complex systems (4.6) and for the isolation of the impact of individual sources of error (4.8). This research has been built on a new perspective on the differences between error and uncertainty (4.2).

The new decision making interface used a variety of techniques to present a large volume of information in an intuitive manner. This was accomplished by creating new visualizations of error (8.2.4) and by augmenting familiar design visualization views with error (8.2.2, 8.2.3). One key technique was to allow the decision maker to choose what information was presented. This was accomplished by linking concurrent system views (8.2) in a dynamically reconfigurable interface (p. 90) with a variety of accessible user controls (8.2.3, B.1.2-B.1.4). Also, some information was made available purely on-demand through unobtrusive tooltips (p. 91 & 93).

A Gaussian process metamodel was used as an integral part of the fidelity trade environment to accelerate component and system modeling. This required some extensions and modifications to the standard Gaussian process theory including derivation of analytical derivatives of the Gaussian process response (p. 110), use of a lognormal hyper-prior for optimizer control (p. 114), and inversion of the standard hyperparameters to improve optimization behavior (p. 113). An innovative decision making interface was also developed for training, tuning, and exploring the Gaussian process metamodel. This pragmatic interface provides an intuitive mechanism for working with hyperparameters (9.3.5), online decision making for training point addition (p. 125), and multi-dimensional visualization of the metamodel and the training data (p. 124).

A database was used as a core part of the fidelity trade environment to provide comprehensive information management throughout the complex systems design process. The schema implemented in this research is a powerful subset of a versatile data architecture proposed in this research (6.3). This architecture provides for the management of data and metadata in addition to providing mechanisms for information accountability, assurance, protection, and validation.

14.2 Limitations

The fidelity trade environment developed in this document presents a powerful new technique for complex systems design. However, there are some identified limitations to what is presented here. Some of these limitations are specific to this implementation, while others are general to the technique itself. These limitations provide context for understanding the strengths and weaknesses of the technique and also provide a guide for future improvements to the technique. Some of the limitations of the fidelity trade environment that have been identified are discussed below.

The sensitivity approach to propagating error through the system amounts to a linearization and is limited to small errors. This is appropriate for most fidelity trade studies and is required to provide an interactive tradeoff environment. However, this limitation should be kept in mind when using the environment for decision making.

The fidelity tradeoff environment operates within the black box paradigm presented by most complex systems design tools. This prevents components from communicating through any sort of bulk data transfer. While this limitation presents some inconvenience when building a system model, it seldom presents a real obstacle and is common to most systems design techniques.

This implementation of a fidelity tradeoff environment does not support arbitrary connections between components. Information flow between components in a system is assumed to happen when quantities have identical names. Similarly, these quantities are assumed to be of the same units and need no intermediate conversion. The flexibility of building system models could be greatly improved by adding support for arbitrary quantity connections with the option of performing unit conversions at that time.

The database setup interface is incomplete. Some entities are cumbersome to work with and some features are not implemented. However, the setup interface provides the vast majority of the required functionality, and this limitation does not reflect on the utility of the error tradeoff environment.

In many places, periodic polling (typically every ten seconds) is used to check for updates between cooperating programs or threads. Interactivity would be greatly improved if the

polling were replaced with a triggered event. There are also some known software bugs in the fidelity tradeoff environment. Many of these are related to the error checking required to make a truly robust end-user program.

The training process for the Gaussian process metamodel can be very slow with very large data sets. This could be improved by profiling and optimizing the implementation, or by implementing some acceleration techniques outlined in the literature [67]. This limitation provides an obstacle to the use of the Gaussian process on large scale problems but does not reflect on the presented techniques.

The visualization of the training data in high-dimensional problems loses the intuition present for low-dimensional problems. Furthermore, the radius threshold setting becomes too sensitive for filtering the displayed training points. Other filtering approaches may help resolve this problem, possibly including a nearest-n filter. If the display is limited to Gaussian process metamodels, the filter could be based on the magnitude of the covariance function. Highly similar points would be displayed, while dissimilar points would be hidden. This would have the impact of incorporating the knowledge embedded in the hyperparameters into the point filter.

The metamodel interface has no built-in tools for metamodel validation. In addition to other interface enhancements, the Gaussian process interface should have an intuitive mechanism for testing and visualizing the quality of the metamodels beyond a visual inspection of the model and the training data.

The system error visualization environment is slow to respond to some user inputs. This is in part due to the inherent complexity of the task at hand, but also due to some details of the implementation. The interactivity could be improved by profiling and optimizing the code. Furthermore, updates could be performed more intelligently, including not reconverging the system when error levels are changed, or by improving the initial guess of each point in the system explorer based on the expected change in the perturbed quantity.

The visualization tools presented do not scale to truly large problems. The system explorer becomes unreadable if one increases the number of inputs or outputs beyond the limits of the graphical display. This would best be resolved by allowing the user to turn

off the display of less interesting input or output quantities. Similarly, the error breakdown stacked bar graph becomes unreadable if there are too many sources of error. This could be addressed by placing filters on the error sources displayed. These could be a top-n type filter, or a filter based on error type. Alternately, a Pareto ranking of the most significant error sources may be an appropriate solution.

At present, the system error visualization environment uses a static version of the Gaussian process metamodel obtained at program startup. It may be beneficial to make the environment use the dynamic version of the metamodel as provided to the metamodel training interface. However, this functionality should be explicitly optional, because it comes at the risk of providing confusing results to the user.

14.3 Future Work

The conception and development of a new complex systems design technique provides ample directions for future research. Of course, there is much work suggested by the limitations described in the previous section. Some more involved ideas which have become apparent to the author have been described below.

The Gaussian process metamodel interface could be improved with a mechanism to find various local minima in the hyperparameter space and present them for the user to evaluate. This feature would be very useful if the Gaussian process were used to model noisy data sets.

The Gaussian process metamodel could be improved to use analytical derivatives for training if they are available. Similarly, additional covariance functions could be added to support discontinuous, periodic, and variable length scale behavior in the response being approximated. This would also require an intuitive interface to select the best covariance function for the particular problem. Automatic relevance detection may be an appropriate technique to explore to guide the choice of covariance functions.

More kinds of decisions may be enabled by an understanding of the flow of error through a complex system. For example, an investigation of the back-propagation of error through the system may enable the designer to understand the tolerance of the selected design point.

For example, if a design point of 1.0 is selected, it probably means the same thing as a design point of 1.00001. Does it mean the same thing as 1.01? What about 1.1? Or 1.5?

The overall architecture including the database schema have been designed to facilitate future work in the areas of variable fidelity and adaptive metamodeling. These active avenues of research would benefit from the tight integration with existing systems design techniques and the work performed in this document.

There is much research to be done in understanding the role of information throughout the complex systems design process. Research could start with a complete implementation of the data architecture presented in Section 6.3. Tracking the design data and metadata of a distributed team presents many challenges to complex systems design.

It may be possible to development optimization algorithms and design techniques that use an understanding of error throughout the system. The first use may be as an informed stopping criterion, but other more novel uses of this information may be appropriate.

It should be possible to develop a library of fidelity settings corresponding to various component analysis tools. With such a library, it would be possible to develop a fidelity morph matrix to allow designers to choose from discrete tools rather than continuous dials.

APPENDIX A

ERROR STABILITY

The counterintuitive phenomenon of decaying error sources discussed in Section 4.7 was encountered in the transport aircraft example discussed in Chapter 12. The diagonal of the system sensitivity inverse ($\mathbf{M}^{-1}$) matrix at the design point was listed in Table 54. The numbers discussed in this appendix are reported to five decimal places to facilitate inspection of their detailed numerical behavior. This does not imply an extended level of confidence or significance in their precision.

Table 54: System sensitivity inverse diagonal

Quantity	Value
W	0.70491
σ	1.07725
W_f/W	0.70491
θ	0.99862
$S_{wet,wing}$	1.27893
$S_{wet,fuse}$	0.99997
$S_{wet,add}$	1.10643
$C_{D,0\ cr}$	0.80017
$C_{D,0\ sl}$	1.0
e_{cr}	0.98342
e_{sl}	1.0
S_{TO}	1.0
S_{LDG}	1.0
$P_{s,sl}$	1.0
$P_{s,cr}$	1.0

Table 54 reveals interesting insight into the behavior of the complex system. First, note that six of the terms are exactly equal to one. These six output quantities do not participate in coupling. The final four quantities (S_{TO}, S_{LDG}, $P_{s,sl}$, and $P_{s,cr}$) are pure output quantities, i.e. no other quantity depends on them. The other two quantities with diagonals equal to one ($C_{D,0\ sl}$ and e_{sl}) are only depended on by the pure output quantities.

All quantities with a diagonal not equal to one participate in the coupling of the system.

Two of the terms are nearly equal to one (θ and $S_{wet,fuse}$). The first, θ, the cruise temperature ratio is only a very weak function of the vehicle size. In fact, θ should be constant so long as the vehicle cruises in the stratosphere. However, the atmosphere metamodel introduced in Section 12.1.2 has low amplitude wiggles due to the difficulty of fitting a non-smooth function with a function which has infinite continuous derivatives as supplied by this implementation of a Gaussian process. Consequently, $S_{wet,fuse}$ has a nonzero diagonal term because it is a function of θ.

The seven other terms listed in Table 54 are significantly different from one, and participate significantly in the coupling of the system. Of these terms, four are less than one, and three are greater than one. The largest and smallest diagonal terms were selected for further investigation. For each case, a single 1% error source in the corresponding quantity was introduced and propagated through the system. The system level errors resulting from a 1% error source in the takeoff gross weight calculation were listed in Table 55. Similarly, the system level errors resulting from a 1% error source in the wing wetted area calculation were listed in Table 56.

Table 55: Propagated error from 1% source in W

Quantity	Value
σW	0.70491 %
$\sigma\sigma$	0.23544 %
$\sigma W_f/W$	0.18553 %
$\sigma\theta$	3.20937×10^{-4} %
$\sigma S_{wet,wing}$	0.75633 %
$\sigma S_{wet,fuse}$	5.99710×10^{-5} %
$\sigma S_{wet,add}$	0.50407 %
$\sigma C_{D,0\ cr}$	0.39653 %
$\sigma C_{D,0\ sl}$	0.33850 %
σe_{cr}	3.06309×10^{-2} %
σe_{sl}	2.70660×10^{-2} %
σS_{TO}	5.03960×10^{-3} %
σS_{LDG}	2.20047×10^{-2} %
$\sigma P_{s,sl}$	0.14448 %
$\sigma P_{s,cr}$	2.24509 %

Table 56: Propagated error from 1% source in $S_{wet,wing}$

Quantity	Value
σW	0.25997 %
$\sigma\sigma$	0.24243 %
$\sigma W_f/W$	0.16345 %
$\sigma\theta$	3.30466×10^{-4} %
$\sigma S_{wet,wing}$	1.27893 %
$\sigma S_{wet,fuse}$	2.21169×10^{-5} %
$\sigma S_{wet,add}$	0.18590 %
$\sigma C_{D,0\ cr}$	0.38158 %
$\sigma C_{D,0\ sl}$	0.29492 %
σe_{cr}	1.12264×10^{-2} %
σe_{sl}	9.87298×10^{-3} %
σS_{TO}	8.95290×10^{-3} %
σS_{LDG}	2.233383×10^{-3} %
$\sigma P_{s,sl}$	0.121507 %
$\sigma P_{s,cr}$	2.149289 %

As expected, the error in the output quantity corresponding to the error source in each case is equal to 1% of the corresponding term of the diagonal of the $\mathbf{M}^{-1}$ matrix. In this case, the system tends to shrink errors in the takeoff gross weight calculation, while it tends to amplify errors in the wing wetted area calculation.

APPENDIX B

SYSTEM INTERFACE

B.1 System Error Visualization Environment Interface

Upon program launch, the program presents the user with a series of choices via a GUI as shown in Figure 117. First, the user chooses a database definition file through a familiar file open dialog. Once a database has been selected, the database is queried for a list of system studies, and the list is presented to the user for selection. Then, for each component in the system study, the database is queried for a list of metamodels. These lists are presented to the user for selection in turn.

B.1.1 System Explorer Ranges

As depicted in Figure 118, the input and response axis ranges and the input hairline values can be changed by double clicking on the axis labels.

B.1.2 Input Control Tab

The system input control tab allows the user to change each of the the input values for the point of interest through a control panel for each input. The control panel allows value control through a slidebar or by changing the numeric value directly. The input ranges can be changed by double clicking on the control background as shown in Figure 118. If there are too many input quantities to be displayed at one time, a vertical scrollbar will appear to make all of the inputs accessible.

B.1.3 Error Control Tab

The error source control tab can be selected to allow the user to change, add, and remove sources of error. The error control tab and its workflow have been depicted in Figure 119; most of the system interface has been cropped from the background to clarify this figure.

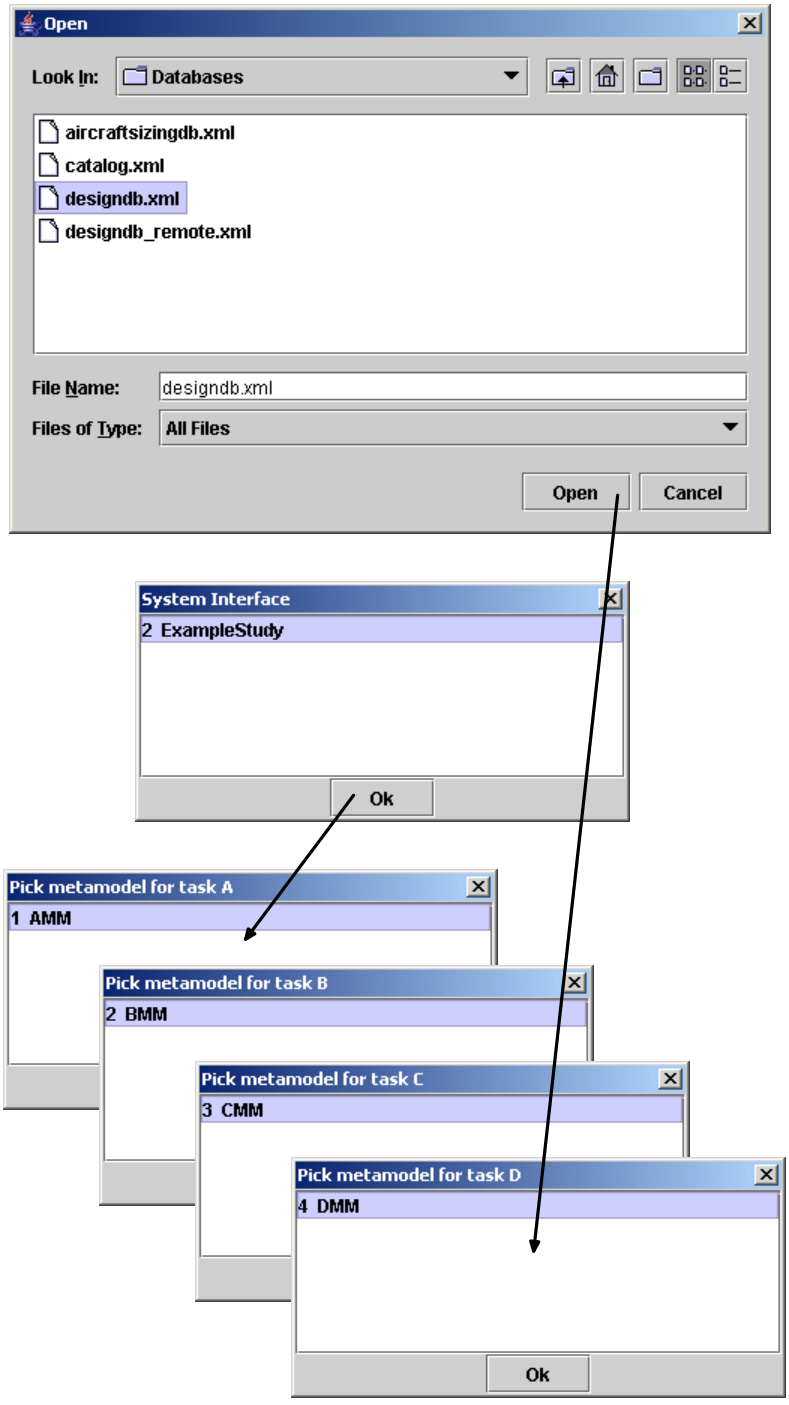

Figure 117: System interface startup

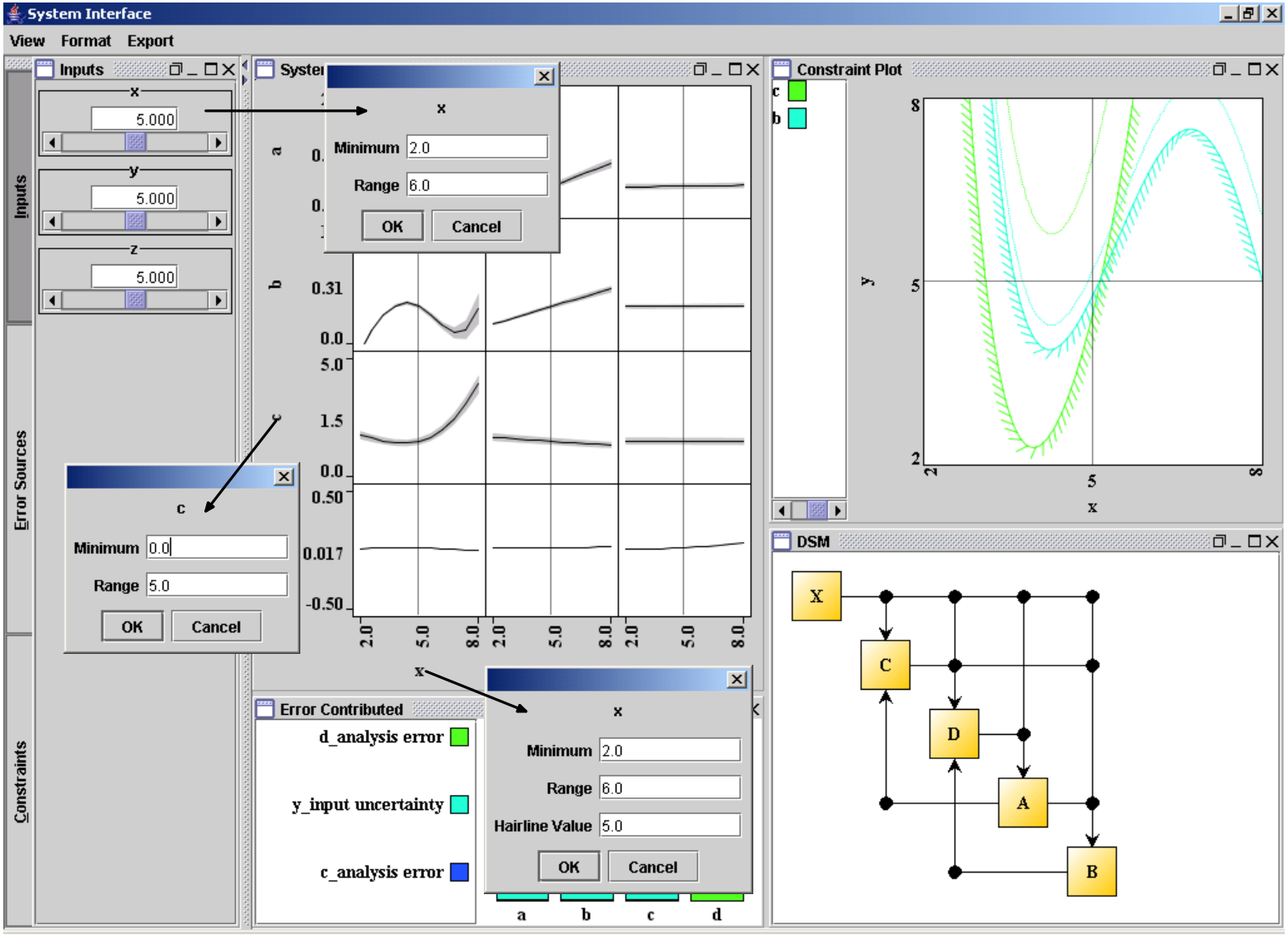

Figure 118: System explorer and input tab range controls

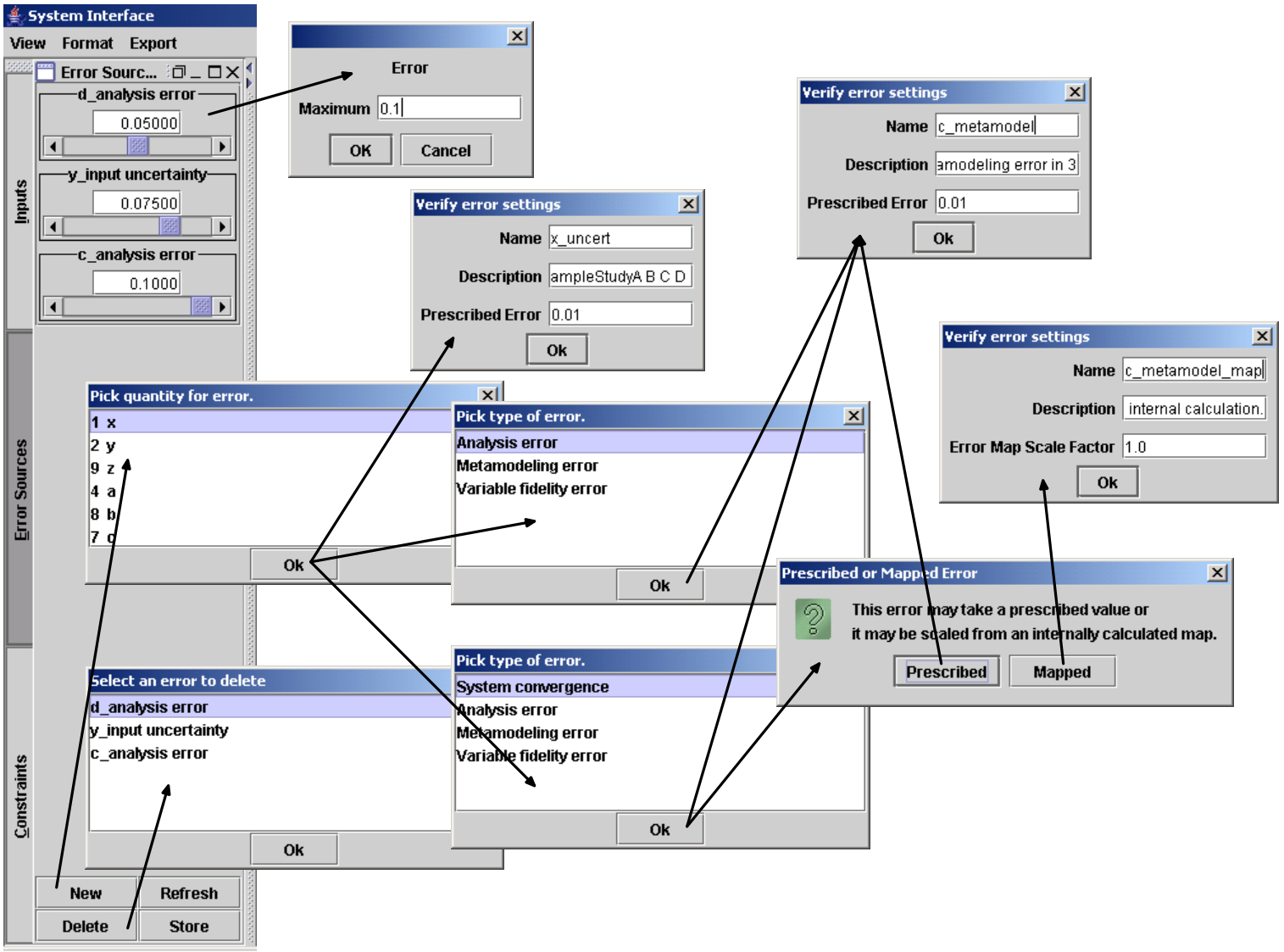

Figure 119: Error source control tab

At the top of the error control tab, there are a series of control panels to interface with each error source. The control panel allows value control through a slidebar or by changing the numeric value directly. The error ranges accessible through the slidebar can be changed by double clicking on the control background as shown in Figure 119. If there are too many error sources to be displayed at one time, a vertical scrollbar will appear to make all of the error sources accessible.

At the bottom of the error control tab, there is a series of buttons for controlling the error sources, and there are buttons named *New*, *Delete*, *Refresh*, and *Store* for operating on an error source. These behave as expected: *New* spawns a series of interactive dialogs to create a new error source; *Delete* spawns a dialog to delete an error source; *Refresh* retrieves the error settings from the database; and *Store* stores the error settings in the database.

When the user clicks the *New* button, he is presented with a series of dialogs to guide the user through creating a new error source. Despite the apparent complexity of this process shown in Figure 119, adding a new error source is quite straightforward. First, the user is presented with a dialog to select which quantity he wishes to apply the error source to; error can be added to any quantity involved in the system. The next dialog presented depends on the type of quantity selected by the user. Mathematically, every error source is treated identically. Differentiation between types of error sources is only done to associate the error source with the appropriate entity in the database; that way, when a component of the system model is reused for another study, the existing error sources are automatically applied.

If the user selects an input quantity, he is presented with a dialog to name, describe, and specify an initial error level for the quantity. Appropriate default values are automatically suggested. Input errors can only be attached to the system study.

If the user selects a quantity calculated by the system, he is presented with a dialog to choose the error type. At least three error types are possible: analysis error, metamodeling error, and variable fidelity error. These three error types correspond directly to the three task implementors specified in the database schema. If the quantity participates in system feedback, a fourth error type corresponding to system convergence error is possible.

It is important to note that this error classification has nothing to do with the root causes of error and it does not imply that any type of error can not fit into this framework. Instead, this classification merely provides a bookkeeping system by which error sources are tracked in the data repository.

If the user chooses to create a metamodeling error, he is presented with a dialog to specify that error source as prescribed or mapped. A prescribed error source is constant at the prescribed level throughout the design space. A mapped error can vary throughout the design space according to an error map supplied by the component. Once the user has selected to create a prescribed or mapped metamodeling error, he is presented with a dialog to name, describe, and specify an initial error level for the quantity. Appropriate default values are automatically suggested.

If the user chooses to create any other type of error, he is presented with a dialog to name, describe, and specify an initial error level for the quantity. Appropriate default values are automatically suggested.

When the user clicks the *Delete* button, he is presented with a list of error sources applied to the system as shown in Figure 119. The user can close the dialog or choose an error source to delete.

B.1.4 Constraint Control Tab

The constraint control tab can be selected to allow the user to change, add, and remove constraints, and to and to change the view of the design space displayed in the constraint diagram. The constraint control tab and its workflow have been depicted in Figure 120; most of the system interface has been cropped from the background to clarify this figure.

At the top of the constraint control tab, there are a series of control panels to interface with each constraint. The control panel allows value control through a slidebar or by changing the numeric value directly. The control panel also allows the user to select whether the feasible region is greater than or less than the constraint value. The constraint ranges can be changed by double clicking on the control background as shown in Figure 120. If there are too many error sources to be displayed at one time, a vertical scrollbar will appear

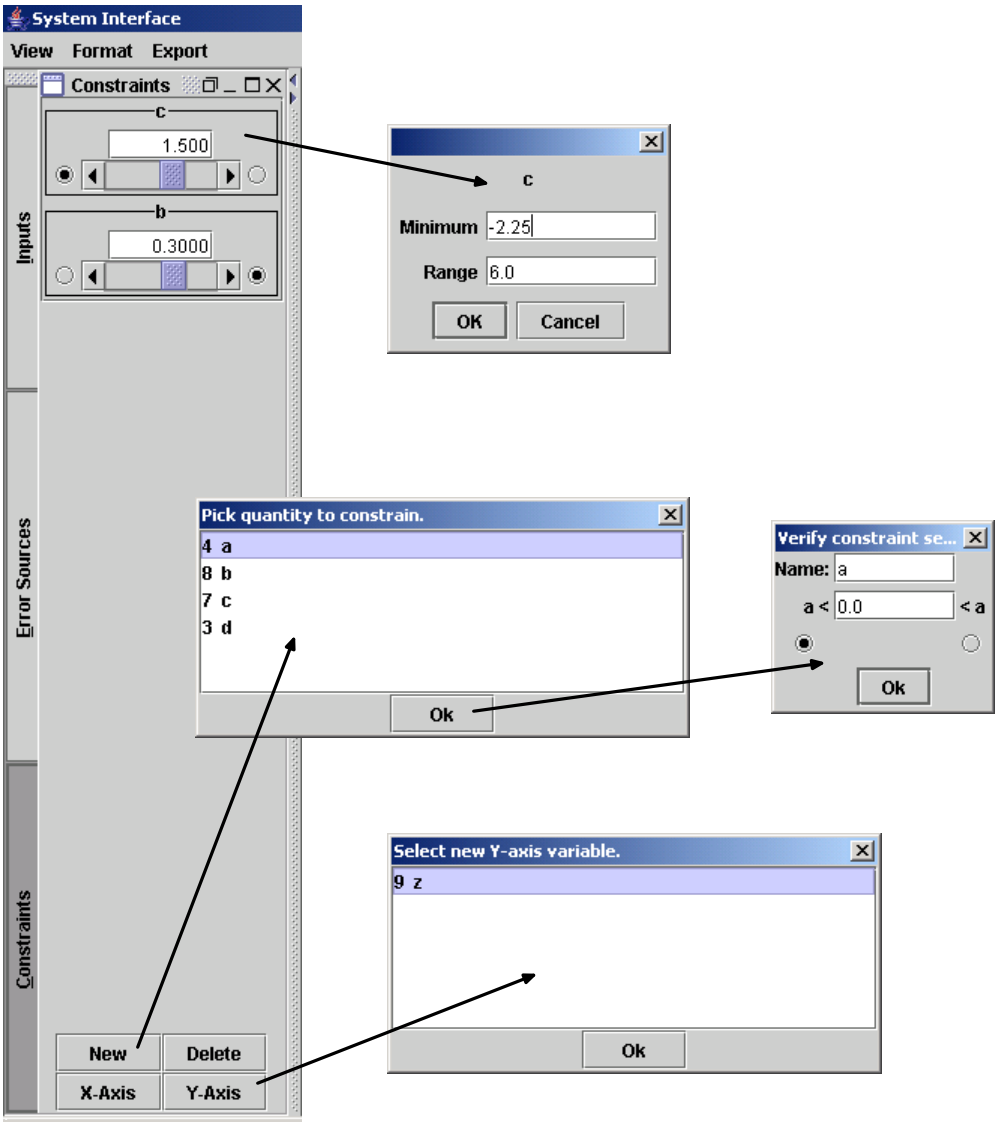

Figure 120: Constraint control tab

to make all of the constraints accessible.

At the bottom of the constraint control tab, there is a series of buttons for controlling the constraints, there are buttons to create a *New* or *Delete* a constraint. These behave as expected: *New* spawns a series of interactive dialogs to create a new constraint; *Delete* spawns a dialog to delete a constraint. There are also buttons to change the X and Y axis of the constraint diagram. Constraints are not stored in the database.

When the user clicks the *New* button, he is presented with a series of dialogs to guide him through creating a new constraint. The process is shown in Figure 120. First, the user is presented with a dialog to select which calculated quantity he wishes to apply the constraint to; constraints can be added to any quantity calculated by the system. The next dialog presented allows the user to set an initial value for the constraint and to specify whether the feasible region is greater than or less than the constraint value.

When the user clicks the *X-Axis* or *Y-Axis* button, he is presented with a list of system input variables not currently used as a constraint diagram axis as shown in Figure 120. The user can close the dialog or choose a quantity to use for the selected axis.

When the user clicks the *Delete* button, he is presented with a list of constraints applied to the system as shown in Figure 120. The user can close the dialog or choose a constraint to delete.

B.1.5 Menu Bar

Across the top of the system interface, there is a standard menu bar for accessing additional functionality. The pull-down menus are labeled: *View*, *Format*, and *Export*.

The *View* pull-down menu presents a list of selectable windows as shown in Figure 121. De-selecting a window causes that window to be minimized. Selecting a minimized window causes it to be restored.

The *Format* pull-down menu presents a list of dialogs which may be used to alter the appearance of the system interface as shown in Figure 122. The *Font...* menu option presents a familiar font selection dialog; changing the font impacts all of the system views.

Selecting the *Prediction Profiler...* option brings up the dialog depicted in Figure 123

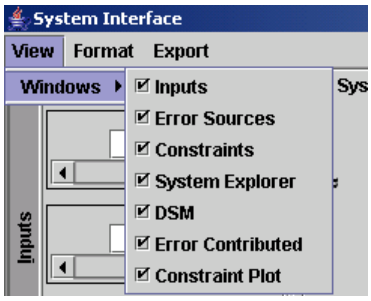

Figure 121: View pull-down menu

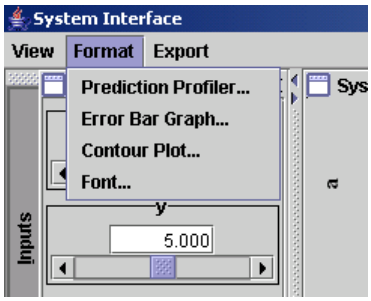

Figure 122: Format pull-down menu

which allows the user to alter the format of the system explorer. All format options required to customize the look of the display are presented, including margins, and line thicknesses.

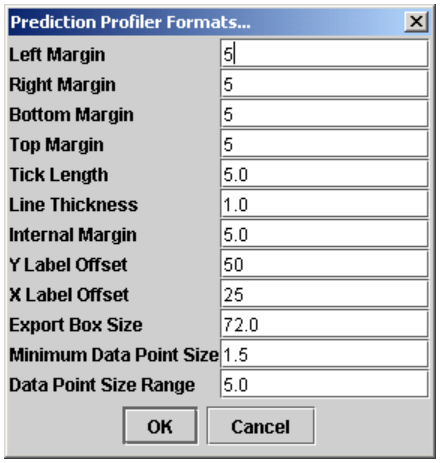

Figure 123: System explorer format menu

Similarly, selecting the *Error Bar Graph...* option brings up the dialog depicted in Figure 124; and selecting the *Contour Plot...* option brings up the dialog depicted in Figure 125.

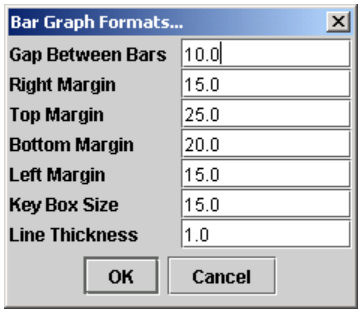

Figure 124: Error bar graph format menu

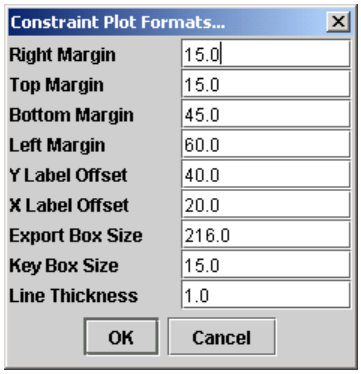

Figure 125: Constraint diagram format menu

The *Export* pull-down menu, as shown in Figure 126, presents a list of dialogs which may be used to produce high quality output from the system interface. The export dialog would be familiar to any computer user and is not depicted here. The *Data dump...* dialog is slightly different; it presents a familiar file save dialog to allow the user to select a file in which to save the state of the system explorer. This includes the point of interest, error, and constraint settings as well as the error breakdown at the point of interest.

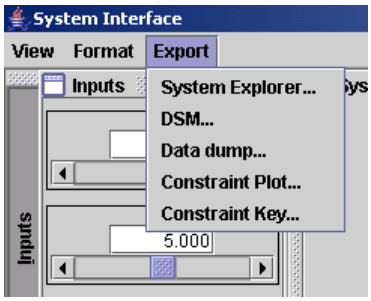

Figure 126: Export pull-down menu

APPENDIX C

METAMODEL INTERFACE

Upon either mode of program launch, the program presents the user with a series of choices via a GUI as shown in Figure 127. First, the user chooses a database definition file through a file open dialog. Once a database has been selected, the database is queried for a list of available metamodels, and the list is presented to the user for selection.

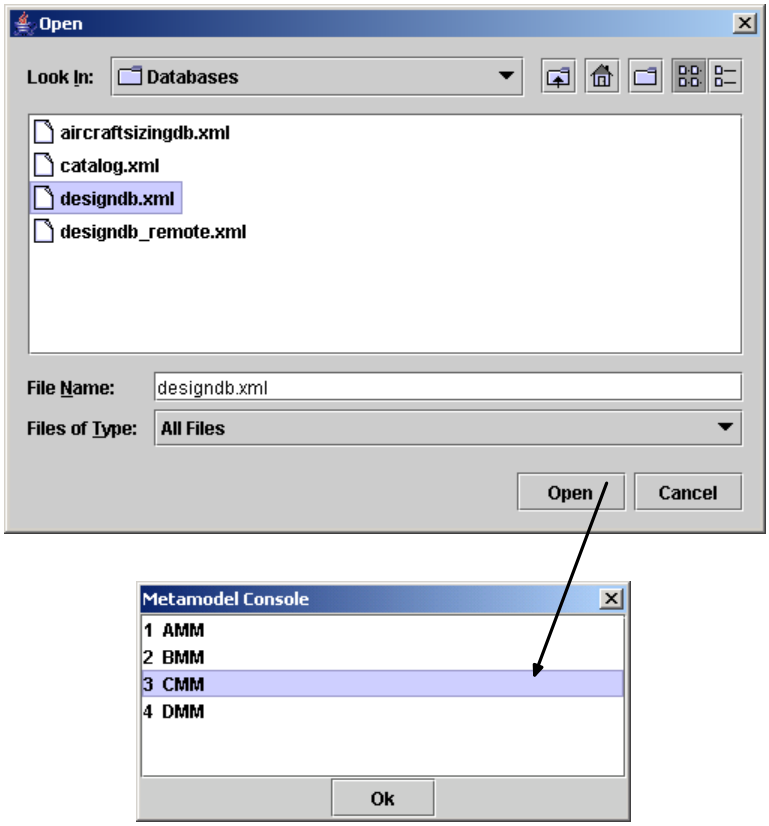

Figure 127: Metamodel startup

C.0.6 Metamodel Explorer

Similar to the system explorer, the input ranges or hairline values can be changed by double clicking on one of the input labels. The dialog box depicted in Figure 128 will appear such

that explicit values may be entered.

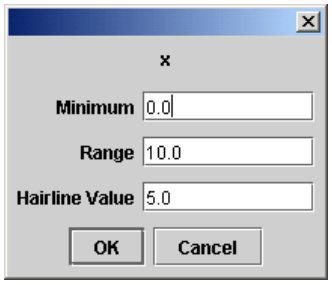

Figure 128: Input change dialog

The input hairline values may also be changed by clicking on a hairline and dragging it to another place within the range of acceptable values. The entire display will be updated upon release.

The response ranges may be changed by double clicking on one of the response labels. This produces the dialog box depicted in Figure 129 in which explicit values may be entered.

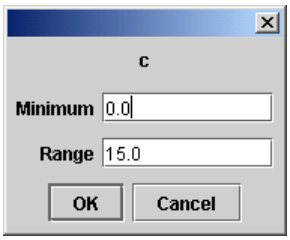

Figure 129: Response change dialog

In the upper left corner of the metamodel explorer window, there is the pull-down menu depicted in Figure 130. This menu has three options, each of which causes another dialog to appear.

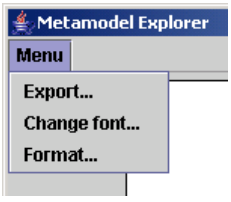

Figure 130: Metamodel explorer pull-down menu

The first menu option, *Export...* brings up a dialog which allows the user to export a

high-quality version of the plot for inclusion in electronic documents. The export dialog would be familiar to any computer user and is not depicted here. An example of the high quality output produced by the export dialog is included as Figure 131. This figure depicts the metamodel in the exact same state as Figure 34.

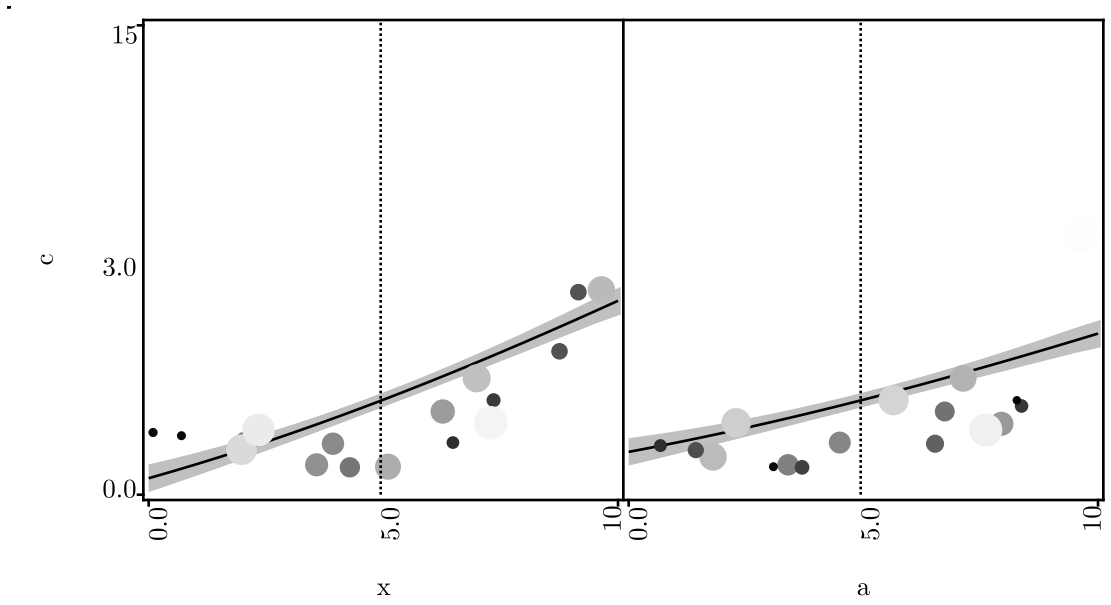

Figure 131: High quality output of metamodel explorer

The second menu option, *Change font...* brings up a dialog which allows the user to change the displayed font to any font currently installed on the computer. The selected font is also embedded in the exported high quality output file. The font choice dialog would be familiar to any computer user, and is not depicted here.

The third menu option, *Format...* brings up the dialog depicted in Figure 132 which allows the user to alter the format of the metamodel explorer. All format options required to customize the look of the display are presented, including margins, line thicknesses, and data point sizes.

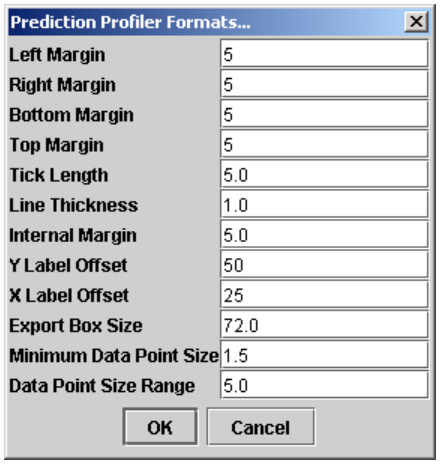

Figure 132: Metamodel explorer format dialog

APPENDIX D

SETUP INTERFACE

Some transition / introduction

D.0.7 Database

The first window presented to the user is used to specify how to find and connect to the database. A screenshot of this window has been included as Figure 133.

In this window, the database *Driver* and Internet *URL* (including the database name) are specified. Fields for the *User name* and *Password* are also provided. Buttons to *Write* or *Read* this information to or from an XML data file are provided. The *Set* button locks in the database choice for the rest of the setup application.

Before a database is locked in, much of the functionality of the setup GUI is not available, and the corresponding buttons are grayed out. When the database is set, a check is performed to make sure the database is accessible and that it has all the appropriate tables defined. If the tables are not defined, the setup GUI will create them at this time.

D.0.8 Server

The next window presented to the user is used to interact with analysis servers and enter the corresponding information into the database. A screenshot of this window has been included as Figure 134.

In the top half of this window, there is a form for specifying all the attributes associated with a server entity in the database. This includes a short *Name* and *Description*, the server location on the network, and appropriate login information. The *Server ID* number, which acts as the primary key, is also listed, but it can not be edited by the user. The ID number is specified by the database when a record is added. The GUI merely displays the ID as appropriate.

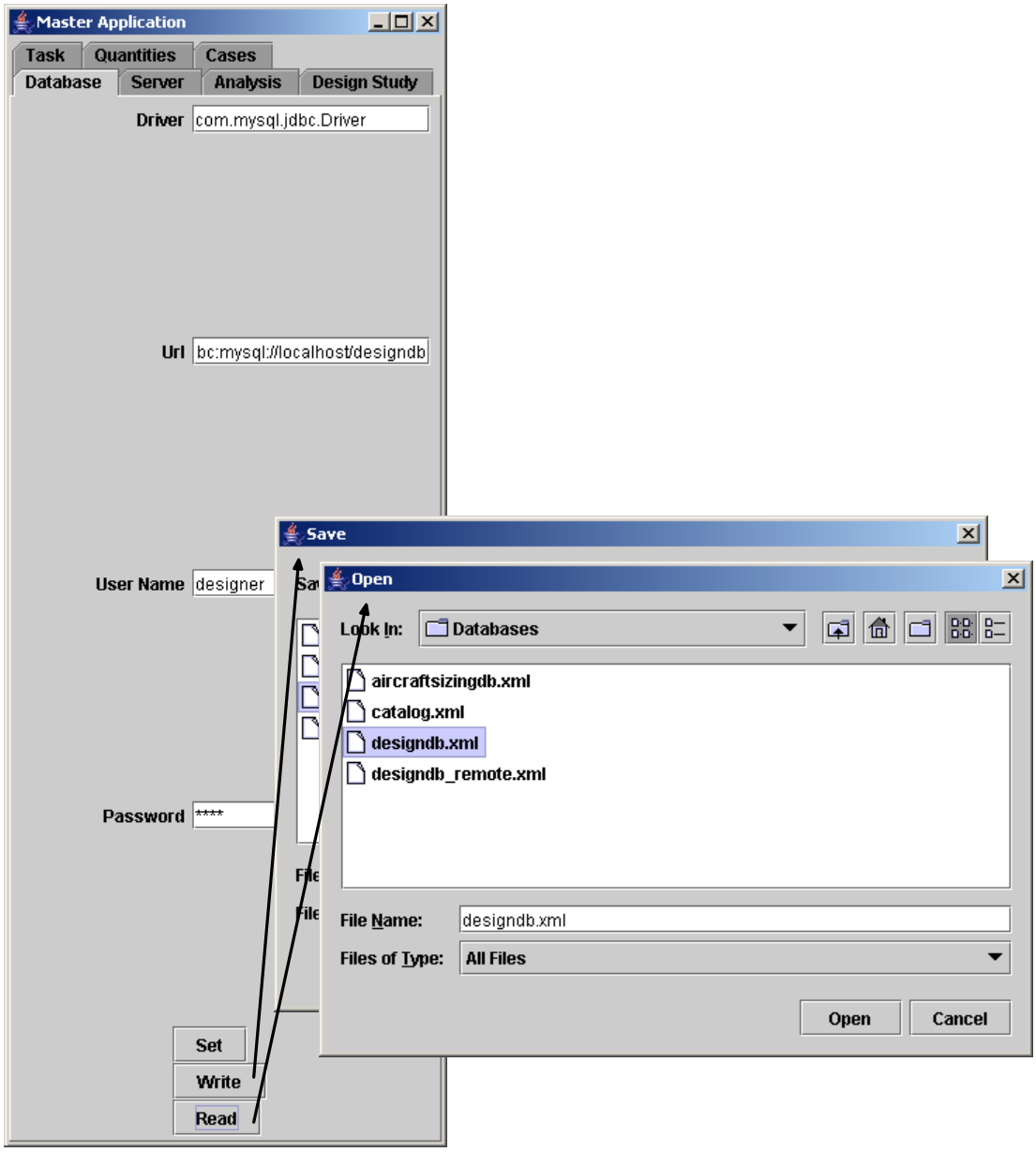

Figure 133: Setup-DB GUI

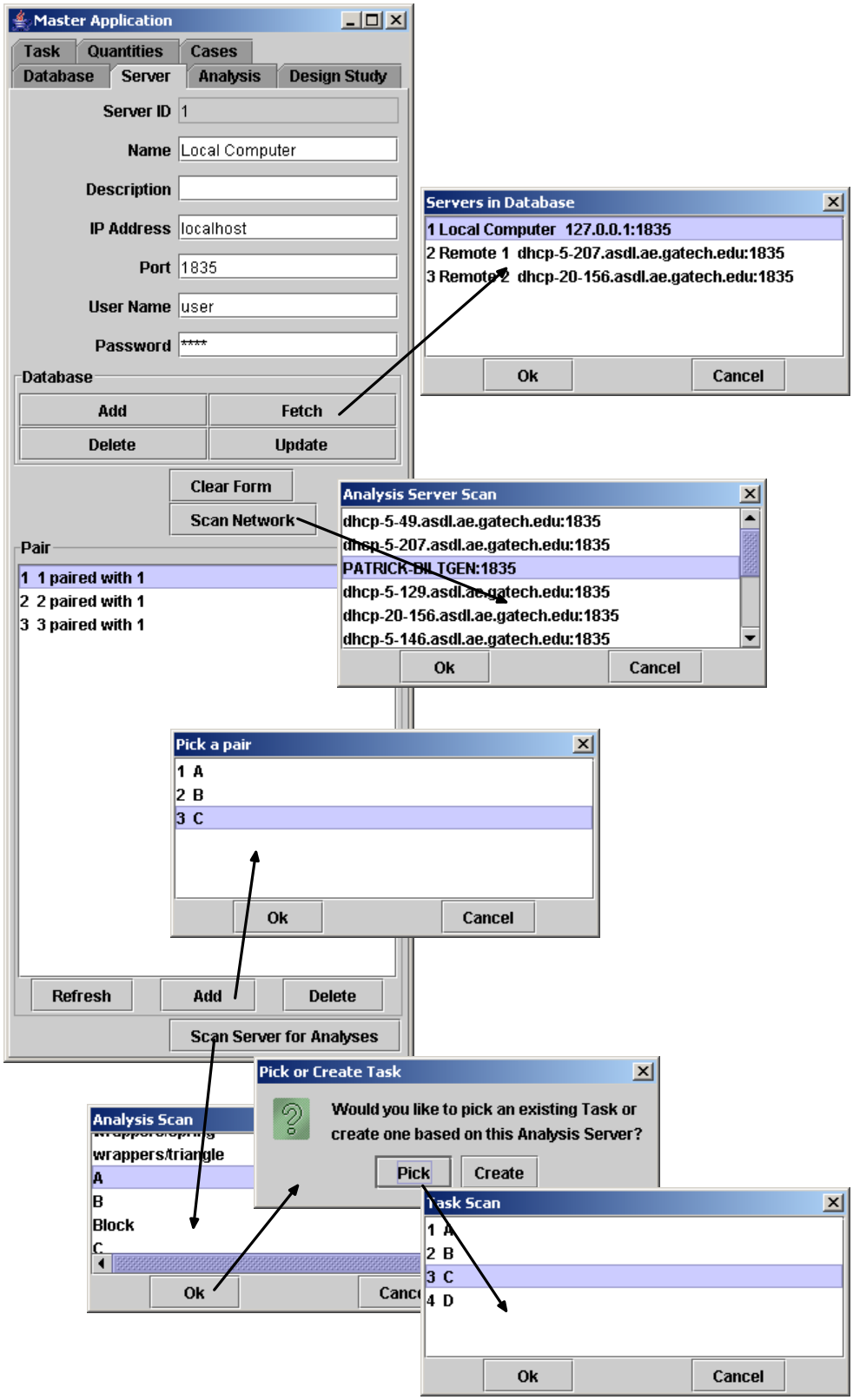

Figure 134: Setup-Server GUI

Immediately below the entry fields, there is a series of buttons for interacting with the database and the form. In the box titled *Database*, there are buttons to *Add*, *Delete*, *Fetch*, and *Update* a record. These behave as expected: *Add* creates a new record with a corresponding ID; *Delete* removes the record with the matching ID; *Fetch* retrieves a record from the database; and *Update* overwrites the record with the matching ID.

The arrow from the *Fetch* button to the *Servers in Database* dialog box in Figure 134 shows what happens when the user clicks the corresponding button. This illustrative technique will be used throughout the discussion of the GUI. To specify which record to retrieve when the *Fetch* button is pressed, a dialog box with a list of all server records is presented to the user. The user may choose a server and click *Ok*, or the user may abort the operation by clicking the *Cancel* button. If the user selects a server, the appropriate information is automatically entered into the form. Most dialog boxes in the setup application have a *Cancel* button that works as expected.

Below the *Database* box, there is a *Clear Form* button which is used to clear all of the fields in the form, including the server ID.

All the tabbed windows corresponding to a database table have a form similar to this for accessing the table's attributes. They all also have the same buttons for accessing the database and clearing the form. The behavior of this standard interface will not be detailed each time it is encountered but when variations are presented they will be noted.

Immediately below the *Clear Form* button there is a *Scan Network* button. When the *Scan Network* button is clicked, a Multicast [51] request for all analysis servers to identify themselves is sent out. All online servers then respond, and a dialog box containing a list of servers on the local network is presented to the user. The user may select a server and click *Ok*. If the user selects a server, the appropriate information is automatically entered into the form. Of course, servers on remote networks may be used, but there is no way to auto-detect them.

This seemingly subtle feature, and others like it, are the reason an interactive setup interface is needed. It allows the user to effortlessly discover and interact with inherently dynamic resources without allowing opportunity for error. Not only may servers be detected

and added, but if a server's dynamic IP address were to change, the user only needs to *Fetch* the record, *Scan Network* for the new address, and then *Update* the database; this may be accomplished with a few mouse clicks, with no typing required.

Below the *Scan Network* button, there is a box entitled *Pair* for creating and manipulating analysis/server pairs. In the window, a list of all existing pairs involving the current server is presented. Pairs are identified and described by ID numbers only. The first pair shown in Figure 134 indicates that pair 1 matches analysis 2 with server 1. Below the window there are three buttons for manipulating the pairings: *Refresh*, *Add*, and *Delete*. The *Refresh* button will scan the database to obtain an up-to-date list of pairings associated with the current server. The *Add* button presents a dialog box with a list of all analyses in the database. If the user selects one and clicks *Ok*, a corresponding analysis/server pair is created. The *Delete* button will delete the selected pairing from the database. Similar interfaces for working with pairings exist throughout the setup program.

The final feature of this window, like the *Scan Network* button, is apparently simple, yet it is possibly the most important feature of the GUI. The *Scan Server for Analysis* button found at the bottom of the window is used to create analysis, task, and quantity records (and the appropriate pairings) based on information provided by an analysis server. When the *Scan Server for Analysis* button is clicked, a list of all analyses available on the server is obtained and presented in a dialog box. When the user selects an analysis and clicks *Ok*, a record for the selected analysis will be added to the database. Of course, the user may *Cancel* without creating a record. When the user chooses an analysis to add to the database, he is then presented with the option to pair the analysis with an existing task, or to create a task based on the information available from the server. If the user clicks *Pick*, they are presented with a dialog box of all tasks in the database to choose which interface the analysis implements. If the user clicks *Create*, records for a new task, all associated quantities, and the appropriate task/quantity pairings are created based on information obtained from the server. This simple step performs most of the work associated with setting up a complex systems design in this schema.

In a typical scenario, when a user needs to model a complex systems study in the

database, most of the job is accomplished through the *Server* window. The user starts by scanning the network and selecting the desired server. Then the user types in a server name and description and a login name and password. The user then adds the server record to the database. Next, the user scans the server for analyses, picks the desired analysis, and lets the interface create the task and accompanying quantities. Finally, the user adds an analysis/server pair corresponding to the freshly created analysis. These last two steps are repeated for all the desired analyses found on the server.

It is because the analysis server itself provides so much information about the available analyses and their interfaces that the setup process is structured bottom-up instead of top-down. A top-down approach (starting with a complex system) to creating the database records would not allow the setup interface to use the information provided by the server to streamline the complex systems modeling process.

D.0.9 Analysis

The next window presented to the user is used to interact with records pertaining to the analysis table in the database. Although this window is fully functional, most of what this window would be used for is accomplished automatically by the *Server* window. A screenshot of this window has been included as Figure 135.

At the top of this window, there is a form for specifying all the attributes associated with an analysis entity in the database. This includes a short *Name* and *Description*, and the *Task ID* of the interface the analysis implements. The *Analysis ID* number is also listed; as before, it is not user-editable. Next to the *Task ID* entry box, there is a *Fetch* button used for the user to pick which interface the analysis implements from those already in the database.

At the bottom of the window, there is a box for creating and manipulating analysis/server pairs. The list and buttons in the box work exactly like those in the *Server* window, except for a change in perspective. The list of pairings is for all matching the current analysis ID, and the *Add* button is used to pick a server from the database with which to create a pairing.

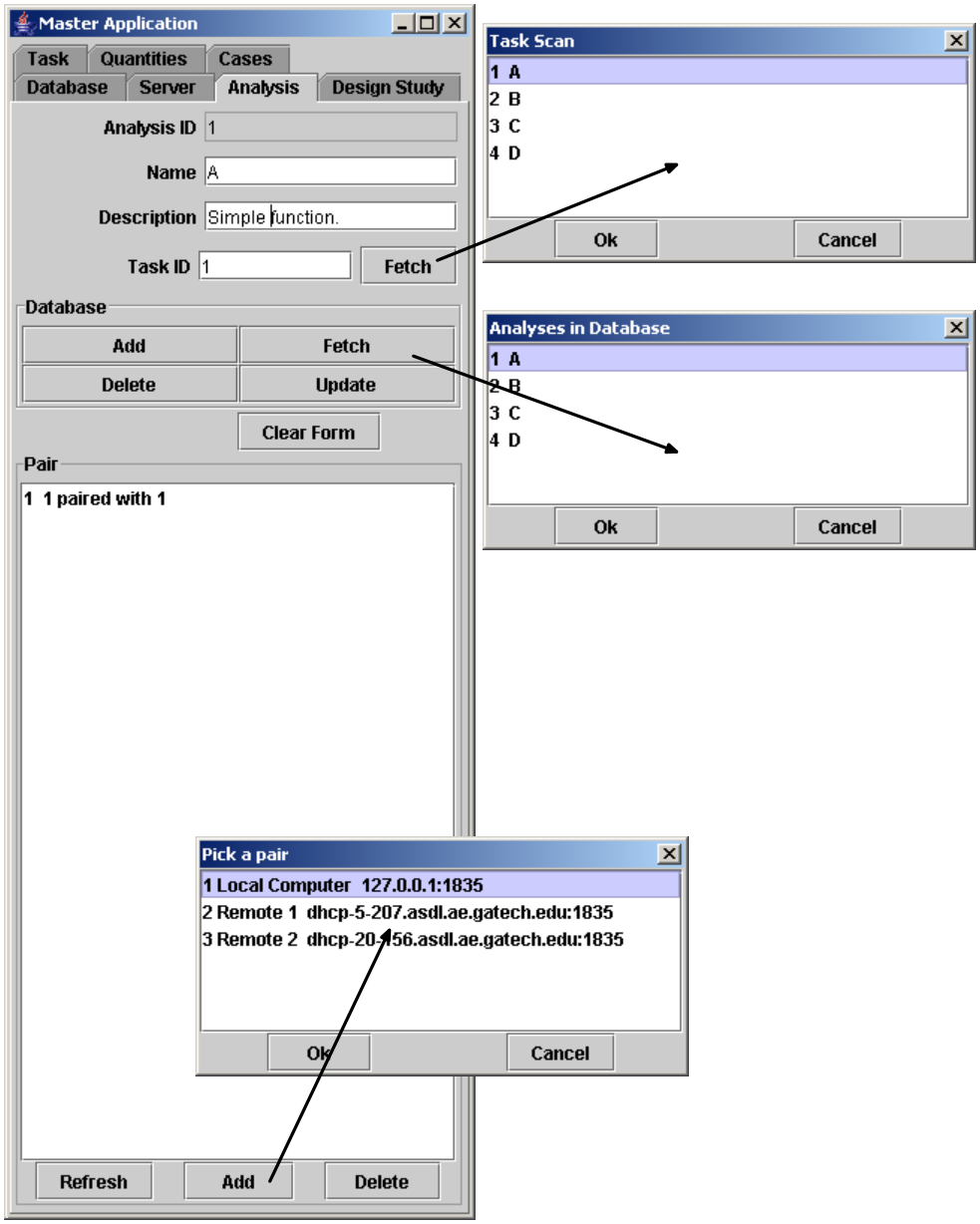

Figure 135: Setup-Analysis GUI

D.0.10 Design Study

The next window presented to the user is used to interact with records pertaining to the study table in the database. A screenshot of this window has been included as Figure 136.

At the top of the form, there are the standard form and accompanying buttons for interacting with the study attributes and database records. Instead of an entry box for the study *Type*, there is a pull-down options menu. At this time, the options for this menu are not used and are generically represented as *type0*, *type1*, etc. The study types are intended to reflect the need for different strategies to handle path based design techniques vs. domain spanning techniques.

Below the form buttons, there is a *Tasks* window and some buttons for working with the tasks associated with a study. This window actually manipulates task/study pairings in a manner similar to the analysis/server pair interface. Instead of displaying a description of the pairing itself, the window simply reports a list of tasks paired with the study. As expected, the *Add* button produces a dialog box with a list of tasks in the database which may be paired with the study. *Delete* and *Refresh* also work as expected.

Below these buttons, there is an interface for working with the quantities associated with the study. Each quantity listed is associated with a study/quantity pairing; variable study/quantity pairs are on the left and objective study/quantity pairs are on the right. The *Refresh* button at the bottom of the interface updates the lists of quantities to match the information in the database.

Variables associated with the design study are displayed in the list on the left of the screen. The extended attributes associated with each variable study/quantity pairing may be edited by selecting the variable, and clicking the *Properties* button. This brings up a dialog box where the *Baseline*, *Minimum*, and *Maximum* values for the variable in the design study may be set. The checkbox to the right of the *Minimum* and *Maximum* entry box set whether bounds are appropriate for this variable. This information is used to indicate the designer's intent in the range of interest to be examined in this study.

There are no buttons to add or delete variables from a design study. This is because the variables for a particular design study are not directly determined by the user; instead they

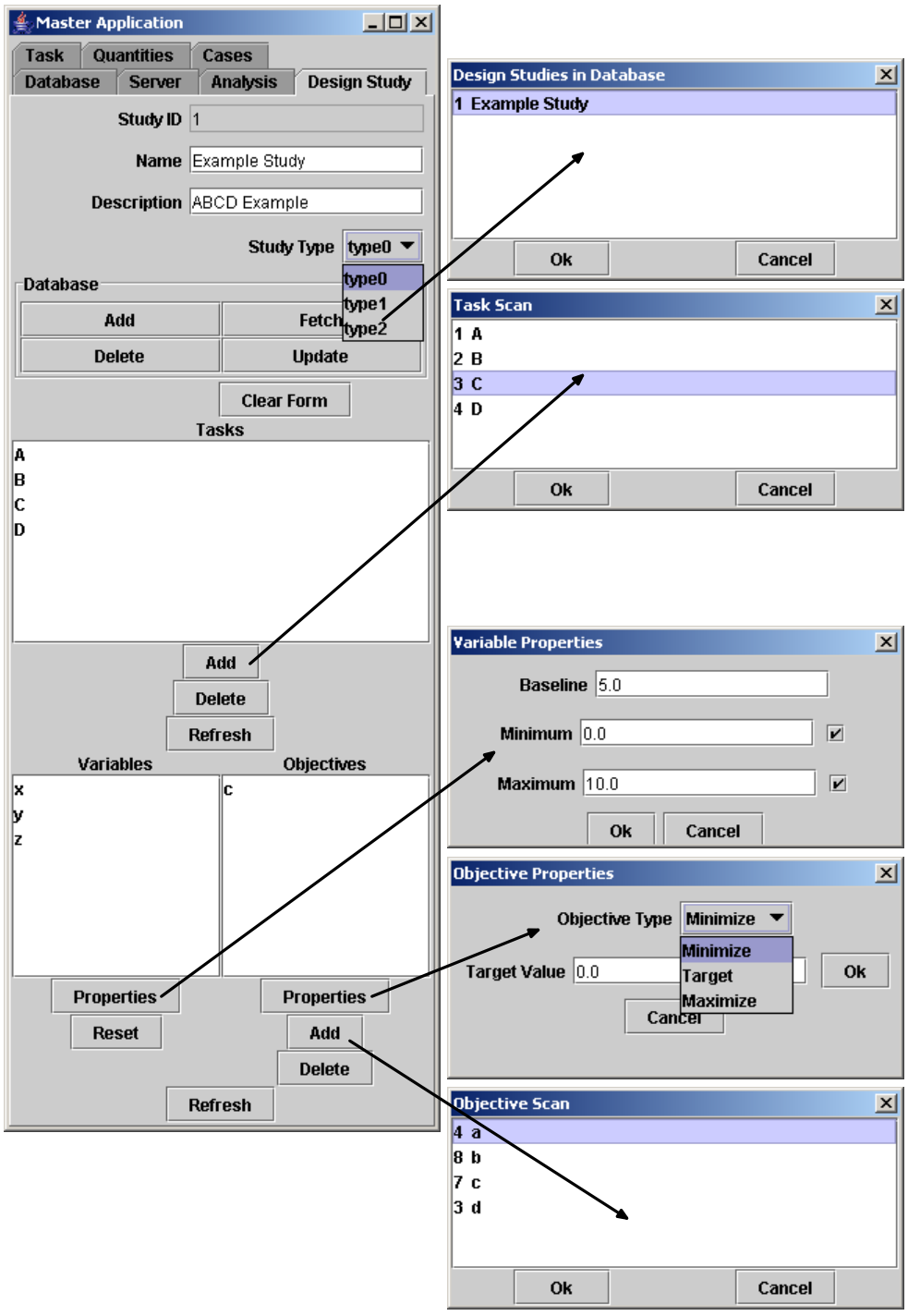

Figure 136: Setup-Study GUI

are implied by the selection of tasks within the design study. Each task has certain input and output quantities and outputs can be used as inputs to another task. All and only those quantities which appear only as inputs to tasks are study variables. This means that once a study is assembled, the user has no input on which quantities are study variables. The list of variables may be built entirely from information stored in the database. The *Reset* button is used to build the list of variables for the study. When the user clicks *Reset*, first all variable study/quantity pairs corresponding to the current study are deleted, then the list of tasks associated with the study is scanned for quantities which appear only as inputs. In doing so, any adjustments to variable properties are lost and must be re-done.

Objectives associated with the design study are displayed in the list on the right of the window. The extended attributes associated with each objective study/quantity pairing may be edited by selecting the variable and clicking the *Properties* button. This brings up a dialog box where the objective *Type* and *Target* value may be set. The objective type is specified by a pull-down options menu. The user may select whether the objective should be *Minimized*, *Maximized*, or driven towards a specified *Target*.

In contrast to study variables, the selection of which (if any) quantities in a design study to use as objectives is very much up to the user. Consequently, *Add* and *Delete* buttons are provided to manipulate the list. When the *Add* button is clicked, a dialog box with a list of all quantities associated with the study's tasks that ever appear as an output is presented. The user may choose outputs from this list. As expected, the *Delete* button removes the selected objective from the database.

D.0.11 Task

The next window presented to the user is used to interact with records pertaining to the task table in the database. A screenshot of this window has been included as Figure 137.

At the top of the form, there are the standard form and accompanying buttons for interacting with the task attributes and database records.

Below the form buttons, there is an interface for interacting with the input and output task/quantity pairs associated with the task. Inputs to the task are displayed on the left.

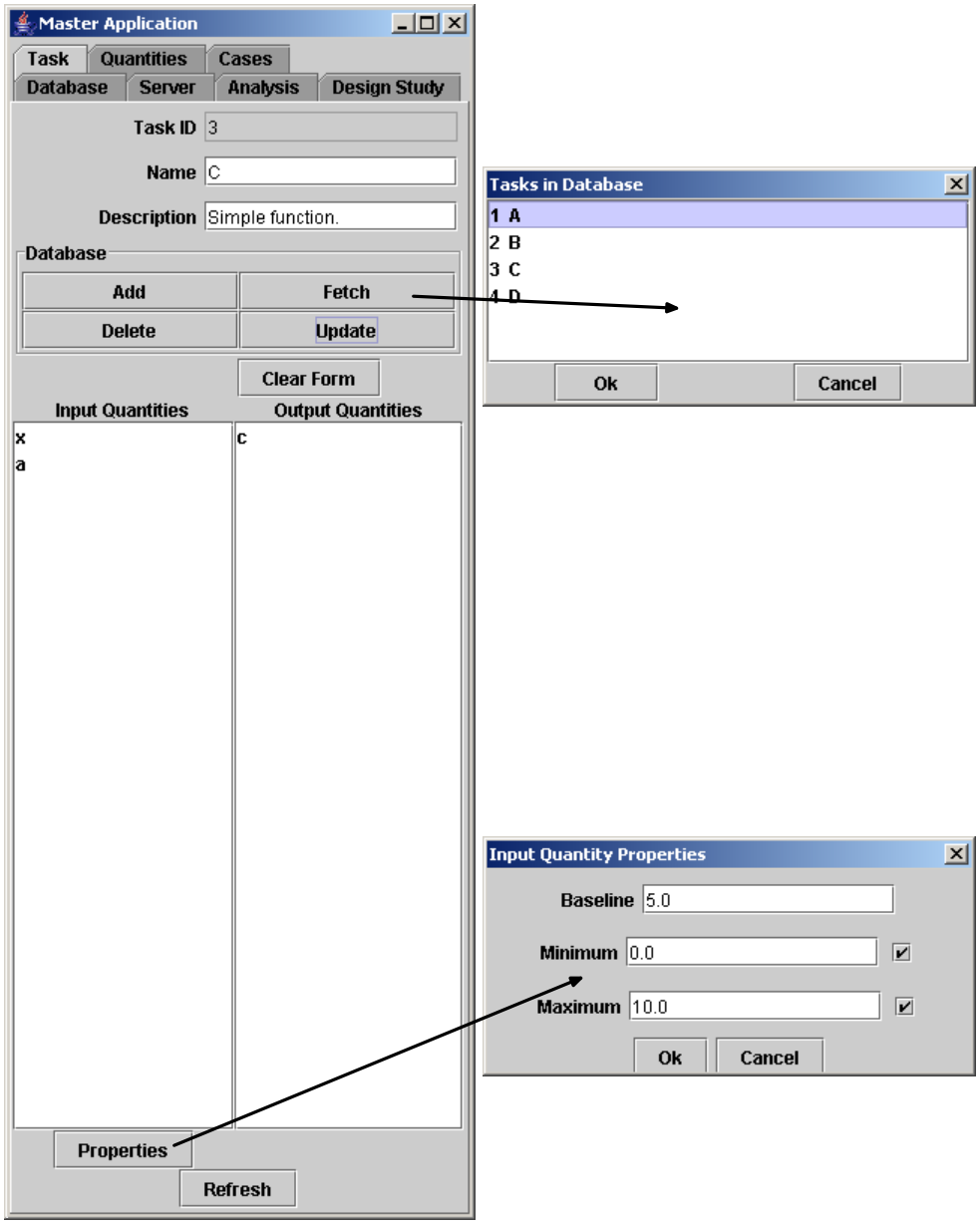

Figure 137: Setup-Task GUI

The *Refresh* button at the bottom of the interface updates the lists of quantities to match the information in the database.

Input quantities are displayed in the list to the left. The extended attributes associated with each input may be edited by selecting the variable, and clicking the *Properties* button below the list. This brings up a dialog box where the *Baseline*, *Minimum*, and *Maximum* values for the variable in the design study may be set. The checkbox to the right of the *Minimum* and *Maximum* entry box set whether bounds are appropriate for this variable. This information is used to indicate any known limits enforced by the task such as physical limitations on a variable (non-negative, etc.) or a range of validity for a theory.

A list of output task/quantity pairs is displayed in the right-hand list. Although there could be some obscure need for the ability to manipulate the list of input or output quantities associated with a task, these lists are best built automatically by the interface through the *Server* window. For this reason, such functionality has been omitted from this window at this time.

D.0.12 Quantities

The next window presented to the user is used to interact with records pertaining to the quantity table in the database. A screenshot of this window has been included as Figure 138.

The majority of the window is consumed by the standard form and accompanying buttons for interacting with the task attributes and database records. The only part of the quantity interface of note is the *Merge* button at the bottom of the window. When a task is created through the *Server* window, quantities corresponding to all of its inputs and outputs are created. Within a design study, these quantities will reappear as inputs and outputs of other tasks. The *Server* interface has no way to recognize that the output from one task is in fact the same quantity as the input to another task. The *Merge* button is used to consolidate duplicate quantities; when clicked, a dialog box is presented with a list of all quantities in the database. When the user selects a quantity from the dialog box, the quantity is merged with the master quantity in the main window. Every reference in

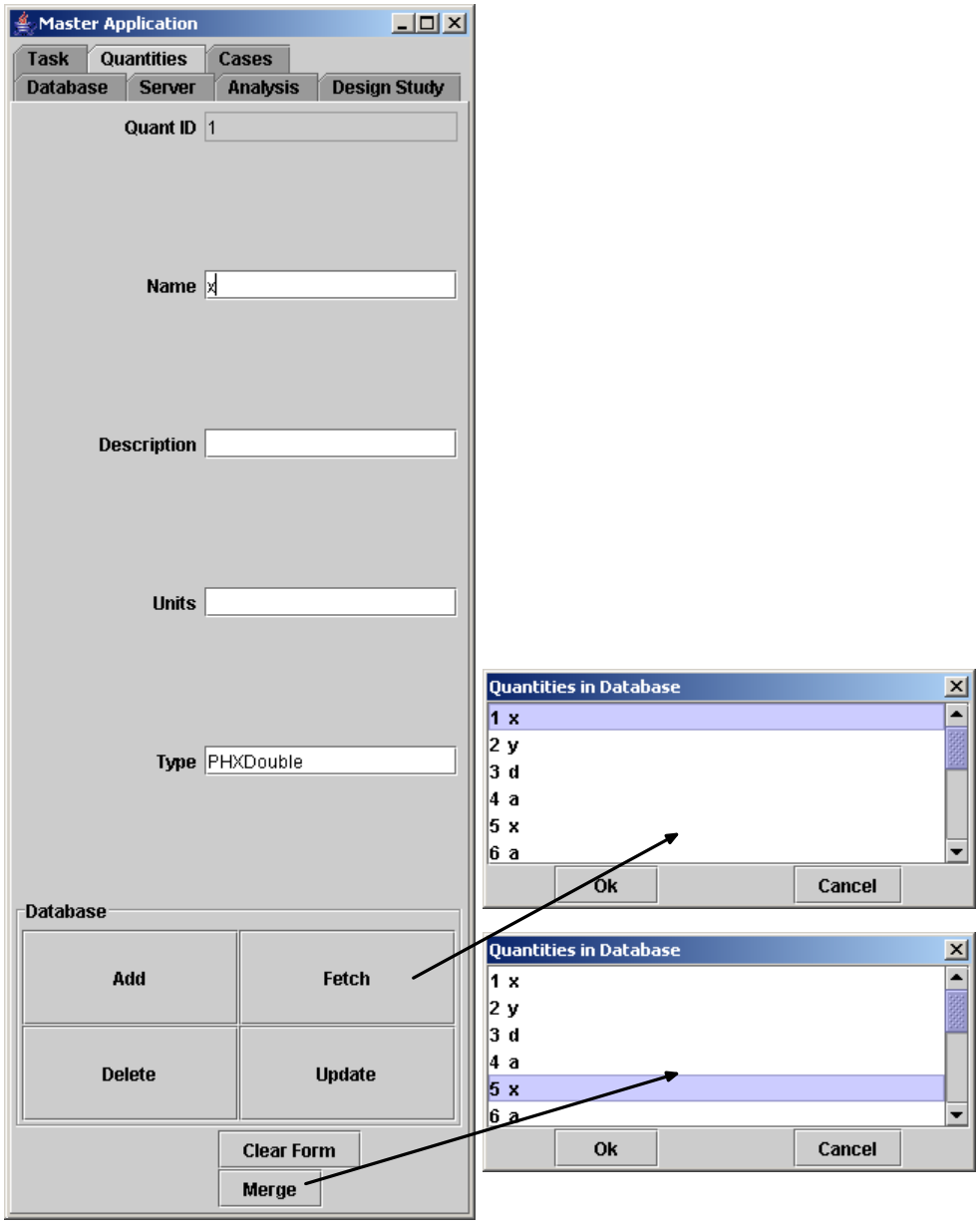

Figure 138: Setup-Quant GUI

the database to the selected quantity is changed to reference the master quantity. Once all references have been changed, the selected quantity is deleted automatically.

D.0.13 Cases

The final window presented to the user is used to interact with records pertaining to the case table in the database. A screenshot of this window has been included as Figure 139.

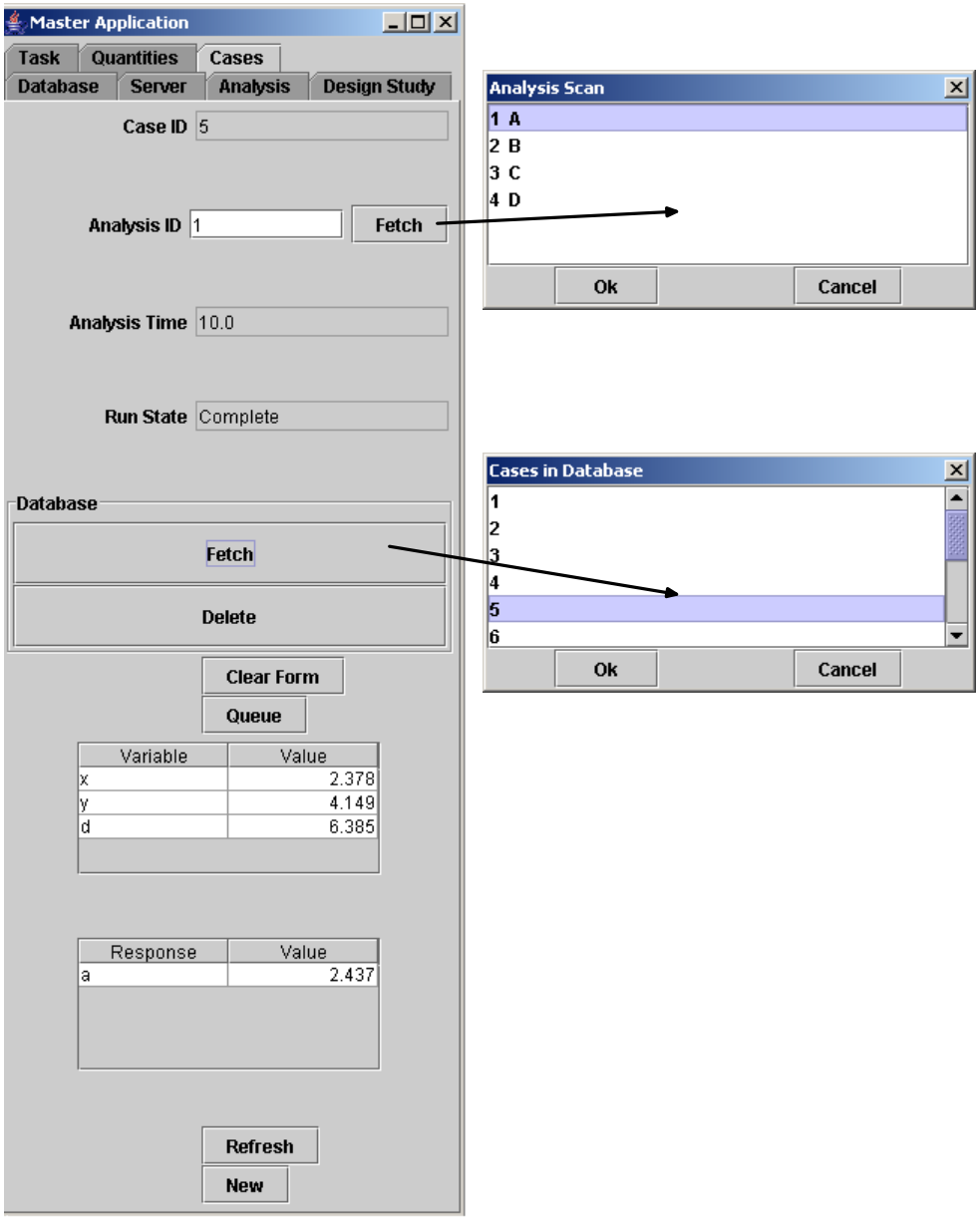

Figure 139: Setup-Case GUI

Unlike most tables in the database, the user does not have much input in the creation

of a case record. Most of the attributes are controlled by the rest of the environment. The user is presented with an entry box and *Fetch* button to indicate which analysis the case is associated with. The *Case ID*, *Analysis Time*, and *Run State* are all controlled by other elements of the environment. The database *Fetch*, *Delete*, and *Clear Form* buttons function as expected. There are no add or update buttons in the traditional sense.

Below the *Clear Form* button there is an interface for creating and queueing cases, as well as setting input values and observing output values. When clicked, the *New* button adds a new case of the appropriate analysis to the database with the *Run State* set to *Prepatory* (-2) and case/quantity pairs are created for all case inputs; the inputs default to the task baseline value. The user may edit the case input values in the spreadsheet below the *Queue* button. Once the desired inputs are set, the *Queue* is used to update the input values for the case and to change the *Run State* to *In Queue* (-1). While a case is running, *Run State* displays *Running* (0). If a case has completed running, the output values will be listed in a spreadsheet above the *Refresh* button and the *Run State* will display *Completed* (1). The *Refresh* button is used to ensure the input and output quantity values displayed in the spreadsheets are up-to-date with the information in the database.

Bibliography

[1] Blumrich, Josef F. Design. *Science*, 168(3939):1151–1154, 1970.

[2] Dieter, George Ellwood. *Engineering Design: A Materials and Processing Approach*. McGraw-Hill Higher Education, 3rd edition, 2000.

[3] Blanchard, Benjamin S. *System Engineering Management*. John Wiley & Sons Inc., 3rd edition, 2004.

[4] Rzevski, George. On the Design of a Design Methodology. In Jacques, Robin and Powell, James A., editors, *Design : Science : Method, Proceedings of the 1980 Design Research Society Conference*, pages 7–17. Design Research Society, Westbury House, 1981.

[5] Cloud, David J. and Rainey, Larry B. *Applied modeling and simulation: an integrated approach to development and operation*. AIAA, Reston, VA, 1989.

[6] Odum, Howard T. and Odum, Elisabeth C. *Modeling for all Scales: An Introduction to System Simulation*. Academic Press, 2000.

[7] Severance, Frank L. *System Modeling and Simulation: An Introduction*. John Wiley & Sons, 2001.

[8] Blanning, Robert W. The Sources and Uses of Sensitivity Information. *Interfaces*, 4(4):32–38, 1974.

[9] Kleijnen, Jack P. C. A Comment on Blanning's "Metamodel for Sensitivity Analysis: The Regression Metamodel in Simulation". *Interfaces*, 5(3):21–23, 1975.

[10] Steward, Donald V. The Design Structure System: A Method for Managing the Design of Complex Systems. *IEEE Transactions on Engineering Management*, EM 78(3):71–74, 1981.

[11] Zeigler, Bernard P., Kim, Tag Gon, and Praehofer, Herbert. *Theory of Modeling and Simulation*. Academic Press, 2nd edition, 2000.

[12] Chung, Christopher A. *Simulation Modeling Handbook: A Practical Approach*. CRC, 2003.

[13] Steward, Donald V. *Systems Analysis and Management: Structure, Strategy and Design*. Petrocelli Books, Inc., New York, NY, 1981.

[14] Tufte, Edward R. *The visual display of quantitative information*. Graphics Press, Cheshire, CT, 1986.

[15] Roache, Patrick J. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, New Mexico, 1998.

[16] Chen, Wei, Allen, Janet K., Tsui, Kwok-Leung, and Mistree, Farrokh. A procedure for robust design: Minimizing variations caused by noise factors and control factors. *ASME Journal of Mechanical Design*, 118:478–485, 1996.

[17] Du, Xiaoping and Chen, Wei. Efficient uncertainty analysis methods for multidisciplinary robust design. *AIAA Journal*, 40(3):545–552, 2002.

[18] Mavris, Dimitri N. and Bandte, Oliver. Efficient uncertainty analysis methods for multidisciplinary robust design. In *Presented at the 2nd World Aviation Congress and Exposition*. Anaheim, CA, 1997.

[19] DeLaurentis, Daniel A. and Mavris, Dimitri N. Uncertainty modeling and management in multidisciplinary analysis and synthesis. In *Presented at the 38th AIAA Aerospace Sciences Meeting and Exhibit*. Reno, NV, 2000.

[20] Wu, Yuin and Wu, Alan. *Taguchi Methods for Robust Design*. ASME, 2000.

[21] Southwest Research Institute. *FPI User's and Theoretical Manual*. San Antonio, TX, 1995.

[22] AIAA. Guide for the verification and validation of computational fluid dynamics simulations. Technical Report AIAA G-077-1998, 1998.

[23] Oberkampf, William L., DeLand, Sharon M., Rutherford, Brian M., Diegert, Kathleen V., and Alvin, Kenneth F. Error and Uncertainty in Modeling and Simulation. *Reliability Engineering & System Safety*, 75:333–357, 2002.

[24] Baird, D. C. *Experimentation: An Introduction to Measurement Theory and Experiment Design*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1962.

[25] Taylor, John Robert. *An Introduction to Error Analysis The Study of Uncertainties in Physical Measurements*. University Science Books, Sausality, California, 2nd edition, 1997.

[26] Oberkampf, William L., Diegert, Kathleen V., Alvin, Kenneth F., and Rutherford, Brian M. Variability, Uncertaintity, and Error in Computational Simulation. In Amano, R. S., Armaly, B. F., Chen, T. S., Emery, A., Liburdy, J. A., Shatto, D. P., Anand, N. K., Blackwell, B., Chu, T. Y., Lage, J., Oosthuizen, O., and Woodbury, K. A., editors, *HTD-Vol. 375-2, ASME Proceedings of the 7th AIAA/ASME Joint Thermophysics and Heat Transfer Conference*, H1137B. 1998.

[27] SAS Institute. *JMP User Guide, Release 6*. Cary, NC, 2005.

[28] Sobieszczanski-Sobieski, Jaroslaw. Sensitivity of Complex, Internally Coupled Systems. *AIAA Journal*, 28(1):153–160, 1990.

[29] Simpson, T. W., Peplinski, J. D., Koch, P. N., and Allen, J. K. Metamodels for Computer-based Engineering Design: Survey and Recommendations. *Engineering with Computers*, 17:129–150, 2001.

[30] Jin, R., Chen, W., and Simpson, T.W. Comparative Studies of Metamodelling Techniques Under Multiple Modelling Criteria. *Structural and Multidisciplinary Optimization*, 23(1):1–13, 2001.

[31] Box, George E. P. and Draper, Norman R. *Empirical Model-Building and Response Surfaces*. John Wiley & Sons, New York, 1987.

[32] McDonald, Robert A. and Mavris, Dimitri N. Formulation, Realization, and Demonstration of a Process to Generate Aerodynamic Metamodels for Hypersonic Cruise Vehicle Design. AIAA-2000-01-5559. AIAA World Aviation Congress, San Diego, CA, 2000.

[33] Koehler, James and Owen, Art. Computer experiments. In Ghosh, S. and Rao, C.R., editors, *Handbook of Statistics, 13: Design and Analysis of Experiments*, pages 261–308. North-Holland, 1996.

[34] Bentley, Jon Louis. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[35] Burden, Richard L. and Faires, J. Douglas. *Numerical Analysis*. Thomson Brooks/Cole, 6th edition, 1996.

[36] MacKay, David J. C. Introduction to Gaussian processes. In Bishop, C. M., editor, *Neural Networks and Machine Learning*, NATO ASI Series, pages 133–166. Kluwer Academic Press, 1998.

[37] Gibbs, M. N. and MacKay, D. J. C. Efficient implementation of Gaussian Processes for Interpolation, 1996. `http://www.inference.phy.cam.ac.uk/mackay/abstracts/gpros.html`.

[38] Bailer-Jones, C. A. L., Sabin, T. J., MacKay, D. J. C., and Withers, P. J. Prediction of deformed and annealed microstructures using bayesian neural networks and gaussian processes. In *Australasia-Pacific Forum on Intelligent Processing and Manufacturing of Materials*. 1997.

[39] Engel, Yaakov, Szabo, Peter, and Volkinshtein, Dimitry. Learning to control an octopus arm with gaussian process temporal difference methods. In Weiss, Y., Scholkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18*, pages 347–354. MIT Press, Cambridge, MA, 2006.

[40] Sinz, Fabian H., Candela, Joaquin Quinonero, Bakir, Gokhan H., Rasmussen, Car. E., and Franz, Matthias O. Learning depth from stereo. In Rasmussen, C. E., Buelthoff, H. H., Giese, M. A., and Scholkopf, B., editors, *Pattern Recognition, Proc. 26th DAGM Symposium, LNCS 3175*, pages 245–252. Berlin, 2004.

[41] Beck, Kent. Simple smalltalk testing: With patterns. *Smalltalk Report*, 1994. `http://www.xprogramming.com/testfram.htm` (June 2001).

[42] *Federal Information Processing Standards (FIPS) Publication 180-2, Secure Hash Standard (SHS)*. U.S. DoC/NIST, 2002. `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`.

[43] Phoenix. *Improving The Engineering Process With Software Integration; Integrating Engineering Applications for Design*. Phoenix Integration, Blacksburg, VA, 2003.

[44] Zentner, Jack, DeBaets, Peter W. G., and Mavris, Dimitri. Formulation of an Integrating Framework for Conceptual Object-Oriented Systems Design. SAE-2002-01-2955. 2002.

[45] Fortran. *Information Technology - Programming Languages - Fortran, Second Edition*. ISO Publications Department, ISO/IEC 1539:1991 edition, 1991.

[46] Mathworks. *MATLAB*. Mathworks, 2005.

[47] Kernighan, Brian W. and Ritchie, Dennis M. *The C Programming Language*. Prentice Hall Inc., 2nd edition, 1988.

[48] Stroustrup, Bjarne. *The C++ Programming Language*. Addison-Wesley Professional, 3rd edition, 1997.

[49] Gosling, James and McGilton, Henry. *The Java Language Environment; A White Paper*. SUN Microsystems, 1996.

[50] Gosling, James, Joy, Bill, Steele, Guy, and Bracha, Gilad. *The Java Language Specification*. Addison-Wesley Professional, 3rd edition, 2005.

[51] Mogul, Jeffery. Broadcasting Internet Datagrams. RFC 0919, The Internet Engineering Task Force, 1984.

[52] Atre, S. *Data Base: Structured Techniques for Design, Performance, and Management*. John Wiley & Sons Inc., New York, NY, 1980.

[53] Piattini, Mario G. and Diaz, Oscar, editors. *Advanced Database Technology and Design*. Artech House Inc., Norwood, MA, 2000.

[54] American National Standards Institute. *ANSI X3.135-1992: Information Systems — Database Language — SQL (includes ANSI X3.168-1989)*. ANSI, 1989.

[55] Ellis, Jon, Ho, Linda, and Fisher, Maydene. *JDBC(TM) API Specification 3.0 Final Release*. Sun Microsystems, Palo Alto, CA, 2001.

[56] MySQL. *MySQL Reference Manual Version 5.0.4*. MySQL AB, 2005. `http://dev.mysql.com/doc`.

[57] MySQL. *MySQL Connector/J Documentation Version 3.1.8-stable*. MySQL AB, 2005. `http://dev.mysql.com/doc/`.

[58] Rogers, James L. A knowledge-based tool for multilevel decomposition of a complex design problem. Technical Report TP 2903, NASA, 1989.

[59] Gebala, David A. and Eppinger, Steven D. Methods for analyzing design procedures. In *ASME Third International Conference on Design Theory and Methodology*, pages 227–233. 1991.

[60] Istratescu, Vasile I. *Fixed Point Theory: An Introduction*. Kluwer Boston Inc., 1981.

[61] Suli, Endre. *An Introduction to Numerical Analysis*. Cambridge University Press, New York, 2003.

[62] Bourke, Paul D. Conrec: A contouring subroutine. *BYTE, The Small Systems Journal*, 12(6):143–150, 1987. `http://astronomy.swin.edu.au/pbourke/projection/conrec`.

[63] Kusnetsov, Andrey. *JGui 2.03*, 2005.

[64] Williams, Christopher K. I. and Rasmussen, Carl Edward. Gaussian processes for regression. In *Advances in Neural Information Processing Systems 8*. 1996.

[65] Neal, Radford M. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer, New York, 1996.

[66] MacKay, David J. C. *Information Theory, Inference, and Learning Algorithms*. Cambridge, 2003.

[67] Rasmussen, Carl Edward and Williams, Christopher K. I. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[68] Santner, Thomas J., Williams, Brian J., and Notz, William I. *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer, 2003.

[69] Sollich, Peter and Williams, Christopher K. I. Using the equivalent kernel to understand gaussian process regression. In Saul, Lawrence K., Weiss, Yair, and Bottou, Léon, editors, *Advances in Neural Information Processing Systems 17*, pages 1313–1320. MIT Press, Cambridge, MA, 2005.

[70] Loader, Catherine. Smoothing: Local regression techniques. *Handbook of Computational Statistics*, 2004.

[71] Abrahamsen, Petter. A review of gaussian random fields and correlation functions. Technical report, Norwegian Computing Center, Oslo, Norway, 1997.

[72] Limpert, Eckhard, Stahel, Werner A., and Abbt, Markus. Log-normal distributions across the sciences: Keys and clues. *BioScience*, 51(5):341–352, 2001. `http://stat.ethz.ch/~stahel/lognormal/bioscience.pdf`.

[73] FreeHEP. *FreeHEP Java Library 2.0-snapshot*, 2006. `http://java.freehep.org`.

[74] Csato, Lehel and Opper, Manfred. Sparse online gaussian processes. Technical report, Neural Computing Research Group, Aston University, 2002.

[75] Brathwaite, Ken S. *Relational Theory: Concepts and Application*. Academic Press Inc., San Diego, CA, 1991.

[76] Harrington, Jan L. *Relational Database Design Clearly Explained*. AP Professional, Chestnut Hill, MA, 1998.

[77] Simpson, T. W., Booker, A. J., Gosh, D., Giunta, A. A., Koch, P. N., and Yang, R.-J. Approximation Methods in Multidisciplinary Analysis and Optimization: a Panel Discussion. *Structural and Multidisciplinary Optimization*, 27(5):302–313, 2004.

[78] Phoenix. *Accelerating Product Development Through Grid Computing; Web Services to Bring the Power of Integrated Computer Resources to the Engineering Process*. Phoenix Integration, Blacksburg, VA, 2005.

[79] Roskam, Jan. Methods for estimating drag polars of subsonic airplanes. 1973.

[80] Schlichting, Hermann. *Boundary-Layer Theory*. McGraw-Hill, Inc., 1979.

[81] U.S. Goverment Printing Office. *U.S. Standard Atmosphere, 1976*. Washington, D.C., 1976.

[82] McCullers, Arnie. Flops user's guide, release 6.02. Technical report, NASA Langley Research Center, 2003.

[83] Mattingly, Jack D., Heiser, William H., and Daley, Daniel H. *Aircraft Engine Design*. American Institute of Aeronautics and Astronautics, 1987.

[84] Anderson, John D. *Introduction to Flight*. McGraw-Hill, Inc., 3rd edition, 1989.

VITA

Robert Alan McDonald was born on August 22, 1977 into a military family in Cheyenne Wyoming. Through his primary education, he moved with his family around the country and abroad, attending school in Alabama, Wyoming, Belgium, and Nebraska. Robert graduated from Papillion-LaVista High School in 1995. Robert then attended the University of Missouri-Rolla (UMR), graduating magna cum laude in 1999 with a BS in Aerospace Engineering and a minor in Mathematics. During this time, he spent one summer performing research at UMR on the use of wing tip devices to reduce induced drag, and two summers performing research at NASA Langley Research Center in Hampton Virginia using unstructured CFD tools to model a high lift wing. Robert then attended the Georgia Institute of Technology, obtaining a MS in Aerospace Engineering in 2001 continuing to obtain his Ph.D. in 2006.